UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Procesamiento de imágenes digitales.

Autor/es

IVÁN PÉREZ ARADROS MARTÍNEZ

Director/es

María Vico Pascual Martínez Losa

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

Universidad de La Rioja

Facultad de Ciencia y Tecnología

Procesamiento de imágenes digitales

Autor: Iván Pérez-Aradros Martínez

Tutora: María Vico Pascual Martínez-Losa

Grado en Ingeniería Informática

Logroño - Junio 2017

Resumen

Para determinar la susceptibilidad de una bacteria a algunos antibióticos se realizan antibiogramas. Para analizar estos antibiogramas se utilizan aparatos que automatizan parte del análisis. El problema, es que estos aparatos están fuera del alcance de algunos centros de investigación, como por ejemplo, la Universidad de La Rioja.

La aplicación AntibiogramJ facilita el trabajo a los científicos, permitiéndoles procesar fotografías de antibiogramas para automatizar parte del proceso de análisis. Para implementar esta aplicación se hace uso de OpenCV, un conjunto de librerías para el tratamiento digital de imágenes. En esta memoria se recoge el trabajo realizado para implementar una versión móvil, para dispositivos Android, de AntibiogramJ. Al igual que en la versión de escritorio, también se utilizarán las librerías de OpenCV.

Abstract

Antibiograms are performed to determine the susceptibility of a bacterium to some antibiotics. Different devices that automate part of the analysis are used in some research centers. The problem is that these devices are beyond the reach of some research centers, such as the University of La Rioja.

AntibiogramJ is an application that facilitates the work of scientists, allowing them to process antibiograms pictures and automating part of the analysis process. OpenCV, a set of libraries for digital processing of images, has been used to implement this app. In this paper it is contained the work done to implement an Android mobile version of AntibiogramJ. OpenCV libraries will also be used to implement the mobile app.

Índice

Introducción

Uno de los experimentos que los biólogos de la Universidad de La Rioja realizan es el antibiograma. Este experimento se utiliza para determinar la susceptibilidad de una bacteria a algunos antibióticos. Para realizarlo se prepara una muestra de cultivo puro de una determinada bacteria. Esta muestra es sembrada en una o varias placas de Petri. A continuación, se colocan unos discos que contienen diferentes antibióticos.

Cuando ha pasado un cierto tiempo, alrededor de los diferentes discos de antibiótico se formarán o no halos. Estos halos son debidos a la muerte de las bacterias que están expuestas al antibiótico. La zona en la que la bacteria ha muerto alrededor del antibiótico se llama zona de inhibición. Para poder sacar conclusiones, se debe medir el diámetro de los halos y compararlos con dos valores: el valor de resistencia y el valor de susceptibilidad. Estos valores se pueden obtener de unas tablas EUCAST [1] de las que hablaremos más adelante. Comparando estos valores con la zona de inhibición del antibiótico, el comportamiento de la bacteria con el antibiótico puede ser: susceptible, intermedio, resistente o no disponible.

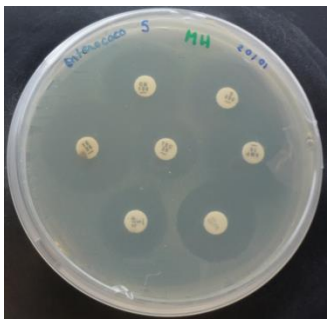

Figura 1: Antibiograma

Hasta no hace mucho, el proceso de medida del diámetro de los halos y su comparación con el valor de resistencia y el valor de susceptibilidad era manual. Existen herramientas que integran el hardware y el software necesario para realizar estas mediciones, pero debido a su elevado coste sólo algunos hospitales o grandes centros de investigación cuentan con ellas. Un grupo de investigación de profesores del departamento de Matemáticas y Computación de la Universidad de La Rioja ha desarrollado una aplicación que permite automatizar este proceso. En la Figura 1 se puede ver como alrededor de cada disco de antibiótico el halo formado es diferente al resto.

Antecedentes

Como punto de partida para el desarrollo de este trabajo tenemos una aplicación de escritorio, AntibiogramJ [3], desarrollada en Java, que permite realizar las mediciones y valoraciones de los antibiogramas de manera automática. Para entender el funcionamiento de esta aplicación es necesario aclarar algunos conceptos.

Un experimento está formado por uno o por varios antibiogramas. Un antibiograma es el que se hace con una misma muestra, pero puede que esta muestra se reparta en diferentes placas de Petri. De cara al desarrollo de la aplicación, esto supone que un antibiograma puede estar formado por más de una imagen.

Esta aplicación permite al usuario gestionar los experimentos mediante un menú. El usuario puede crear antibiogramas en cada experimento e ir añadiendo imágenes a cada antibiograma. Esto último lo realiza a través de un panel de configuración.

Una vez que una imagen ha sido cargada, el programa ofrece diferentes facilidades al usuario para que pueda analizarla. Los pasos que se siguen para el análisis de la imagen son los siguientes:

1. *Datos del aislamiento bacteriano*. El usuario introduce la información del cultivo de bacterias que se ha sembrado en la placa.
2. *Pre-procesamiento de imagen*. AntibiogramJ permite al usuario modificar algunas de las características de la imagen de forma automática o manual. En el paso 2.1 se puede ajustar el tamaño de la imagen a la placa de Petri, como se muestra en la Figura 2. Además, permite cambiar el brillo y el contraste en el paso 2.2.

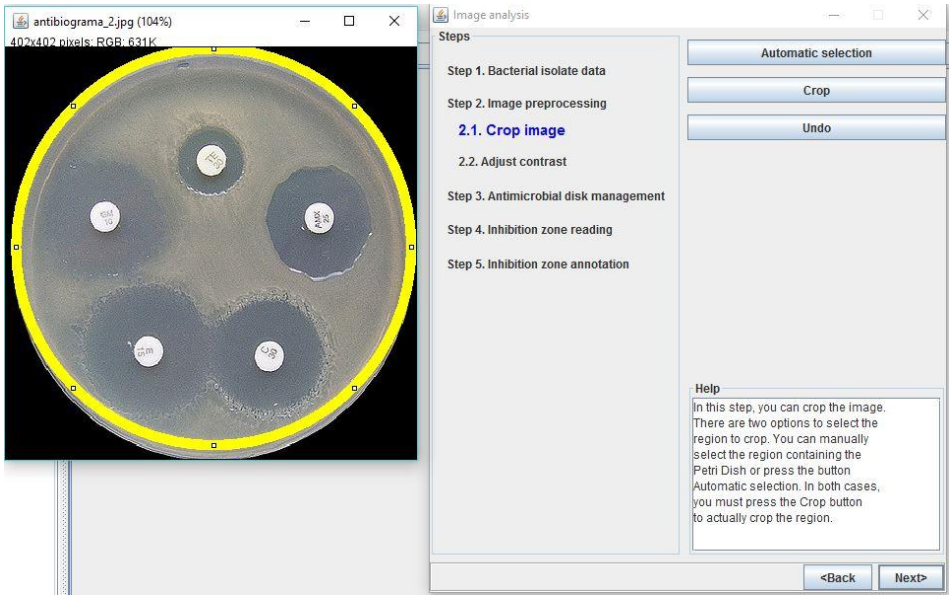

Figura 2: Captura de pantalla de AntibiogramJ.

3. *Gestión de discos antimicrobianos.* La aplicación detecta automáticamente los discos de antibiótico y permite añadir a mano alguno en caso de que no se haya detectado. Esta funcionalidad se controla en el paso 3.1. Para calibrar la escala de la imagen, en el paso 3.2 el usuario puede indicar el tamaño de uno de los discos. El tamaño por defecto es 6mm. Finalmente, en el paso 3.3 la aplicación lee los códigos para identificarlos. En algunos casos, puede ser necesario que el usuario ajuste manualmente la posición de los discos, por lo que la aplicación ofrece opciones para ello. En la Figura 3 se incluye una captura de pantalla de este proceso.

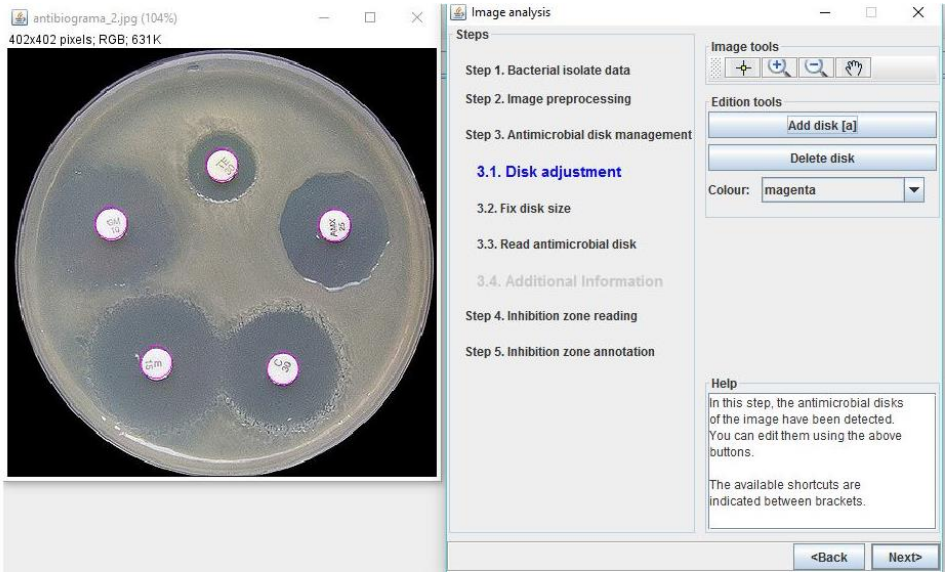

Figura 3: Captura de pantalla de AntibiogramJ

4. *Lectura de la zona de inhibición.* AntibiogramJ detecta automáticamente las medidas de la zona de inhibición. Como en el paso anterior, ofrece al usuario las herramientas para ajustarla manualmente en caso de que sea necesario. Además, muestra las circunferencias correspondientes a los valores de resistencia y susceptibilidad del antibiótico como podemos ver en la Figura 4. La medición de la zona de inhibición con respecto a estos valores, determina si el comportamiento de la bacteria es susceptible, intermedio, resistente o no disponible.

   Para realizar estas valoraciones, la aplicación incluye un mecanismo para cargar tablas EUCAST. Las siglas corresponden a “European Committee on Antimicrobial Susceptibility Testing”, un organismo internacional que entre otras labores, mantiene las tablas que contienen los valores que antes mencionábamos para cada antibiótico y bacteria. Esta información es permanentemente accesible a través del sitio web (http://www.eucast.org/).

   Por defecto, la aplicación AntibiogramJ utiliza los datos del estándar EUCAST v6.0 pero permite utilizar estándares de otras versiones u otros laboratorios. Dado que cada estándar utiliza un formato diferente, la aplicación recoge los datos de ficheros XML que son válidos respecto a un XMLSchema que se ha definido, y que de aquí en adelante denominaremos AntibiogramJXML. La transformación de los datos del estándar correspondiente al AntibiogramJXML se realizará con un programa externo.

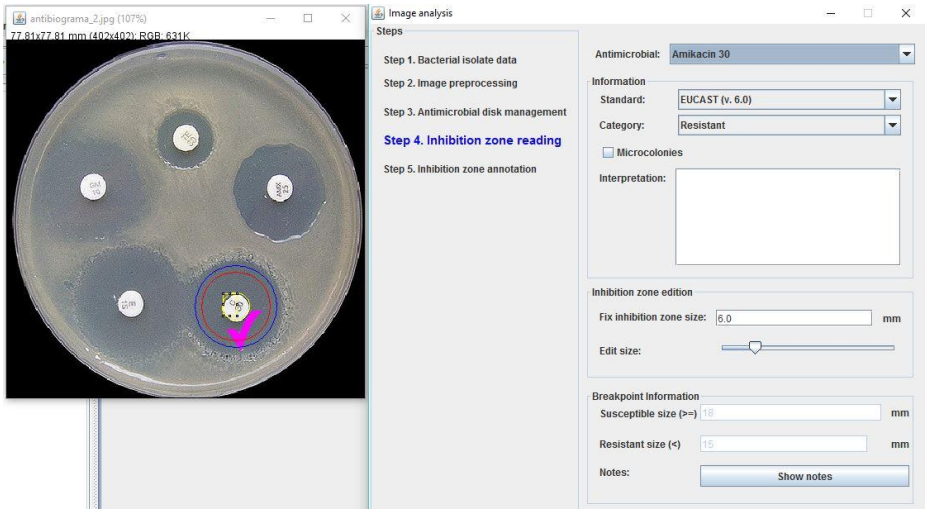

Figura 4: Captura de pantalla de AntibiogramJ

5. *Anotación de la zona de inhibición.* En este último paso la aplicación muestra la imagen incluyendo información anteriormente calculada como el diámetro de cada halo. Esto lo podemos ver en la Figura 5.

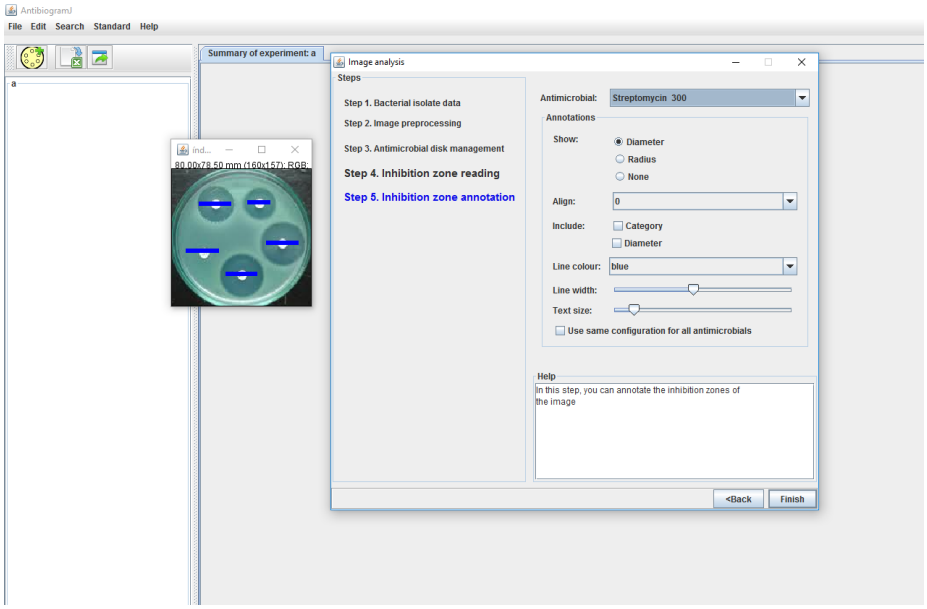

Figura 5: Captura de pantalla de AntibiogramJ

Una vez terminado el análisis de la imagen, éste se podrá ver en el panel del experimento. Además, se pueden guardar los experimentos para trabajar más adelante con ellos.

Para la visión artificial de imágenes esta aplicación está utilizando tecnología OpenCV [1]. Este término proviene de Open Source Computer Vision. OpenCV es una librería de funciones que fue diseñada para ser utilizada en sistemas de tiempo real. Provee a las aplicaciones de visión artificial. Esta librería cuenta con más de 2500

algoritmos que permiten identificar objetos, caras o reconocer escenarios entre otras cosas.

OpenCV está desarrollado en C++ pero tiene interfaces que permiten utilizarlo en C, C++, Python, Java y MATLAB. En la aplicación AntibiogramJ se está utilizando la interfaz para desarrollo en Java. Esta aplicación hace uso de la librería OpenCV para las detecciones automáticas de las zonas de inhibición.

ImageJ [4] es un programa de dominio público para el procesamiento digital de imágenes. Está desarrollado en Java y proporciona extensibilidad vía plugins Java. ImageJ puede mostrar, editar, analizar y procesar imágenes de 8, 16 y 32 bits. AntibiogramJ hace uso de este programa para el procesamiento digital de las imágenes.

Objetivos del TFG

Con la realización de este trabajo, se pretende desarrollar una aplicación móvil que ofrezca una funcionalidad similar a la que ofrece AntibiogramJ en su versión de escritorio. Una de las ventajas de tener una versión móvil de esta aplicación es que permite a los biólogos tomar una foto de la placa con una tableta o con un Smartphone en el propio laboratorio y hacer el análisis *in situ*. Además, si alguna vez realizan algún viaje de investigación, puede resultarles muy útil tanto para hacer nuevos experimentos como para consultar los experimentos que ya habían realizado.

Así, el objetivo principal de este TFG, es desarrollar una versión móvil de la aplicación para escritorio AntibiogramJ. Asociado al desarrollo de esta aplicación móvil surge la necesidad de aprender programación móvil. Como apuntaremos más adelante se quiere que la aplicación soporte cualquier dispositivo que tenga instalado el sistema operativo Android. Luego, en particular, nos centraremos en el estudio de esta tecnología.

Dado que la aplicación tratará con imágenes, es de obligada necesidad el estudio de programas para el tratamiento de las mismas. Siguiendo las líneas de la aplicación de escritorio, será necesario aprender a utilizar la librería OpenCV y el plugin para Java ImageJ. El estudio de los algoritmos que permiten el tratamiento de imágenes será otro de los objetivos añadidos de este trabajo.

Con la realización de este trabajo se persigue también desarrollar algunas de las habilidades que todo director de equipo debe tener. Por tanto, aprender a realizar una correcta planificación de las distintas fases de desarrollo es un objetivo adicional. Saber elegir entre varias opciones la que mejor se ajuste a nuestras necesidades o elegir un diseño adecuado son también metas que se esperan alcanzar.

Objetivos de la aplicación móvil

Como objetivo principal de este trabajo, el diseño y desarrollo de la aplicación móvil, merece un apartado propio. A continuación, se listan los requisitos generales que esta aplicación móvil debe cumplir.

- La aplicación móvil debe dar soporte a todos los dispositivos móviles y tabletas que tengan instalado el sistema operativo Android 4.0 o superior.
- Disponer de una interfaz pensada para pantallas de un tamaño grande (mayores de 7 pulgadas). Para pantallas pequeñas, consideramos que no tiene tanta utilidad debido a la dificultad que conllevaría trabajar con las imágenes.
- Gestionar la carga de imágenes. Éstas serán elegidas desde la galería o tomadas directamente con la cámara de fotos del dispositivo móvil.
- Realizar el procesamiento de imágenes. Se debe permitir tanto el procesamiento automático como la manipulación manual por parte del usuario.
- Almacenar y gestionar la información relativa a antibióticos y bacterias que será necesaria para el correcto uso de la aplicación.
- Permitir la realización de experimentos. Los pasos deben ser similares a los de la versión de escritorio, de forma que a un usuario de esta última no le resulte difícil manejar la versión móvil.
- Gestionar los experimentos. La aplicación debe ser capaz de guardar los datos de un experimento realizado. Además, debe permitir exportar e importar datos de la versión de escritorio.
- La aplicación debe poder cargar automáticamente las tablas del EUCAST a las que nos referíamos en los antecedentes del TFG.
- Debe ser una aplicación robusta. Un fallo nunca puede suponer la pérdida de la información asociada a los experimentos.
- La aplicación debe ser razonablemente eficiente, dando una respuesta que no exceda en más de un 50% el tiempo que emplea la versión de escritorio. Al tratarse de dispositivos móviles, la eficiencia en espacio es un factor que también se debe tener en cuenta.

Al mismo tiempo que decidíamos cuáles serían los objetivos de esta aplicación, se nos iban ocurriendo otros muchos. Debido a la limitación temporal de este trabajo todos ellos no podrán ser alcanzados. En cualquier caso, consideramos interesante su inclusión en esta memoria. Ordenados de mayor a menor prioridad los objetivos secundarios son los siguientes:

- Realizar la gestión de usuarios de la aplicación. Permitir el inicio de sesión con diferentes usuarios.
- Desarrollo de una segunda interfaz pensada para trabajar con dispositivos móviles que tienen una pantalla pequeña con una funcionalidad más limitada.

- Aunque en principio no se pretende que la aplicación de soporte a otros sistemas operativos como iOS o WindowsPhone sería una característica deseable.

Una vez se hayan alcanzado los objetivos prioritarios, se irán realizando los objetivos secundarios. Los posibles problemas que puedan ir surgiendo y la velocidad con la que seamos capaces de encontrar soluciones a ellos marcarán los límites finales de este trabajo. Con casi total seguridad, el objetivo de dar soporte a múltiples sistemas operativos quedará fuera de las posibilidades de este trabajo.

Tecnologías empleadas

Se puede decir que algunas de las tecnologías a utilizar en este trabajo venían casi fijadas por las necesidades. Este es el caso de Java. Dado que la aplicación utilizará OpenCV, programaremos haciendo uso del lenguaje Java. Si bien es cierto que hay otras interfaces de esta librería para programar en Python o en C++, no consideramos recomendable escoger alguno de estos lenguajes pues ello supondría mucho trabajo adicional.

Sin embargo, sería necesario decidir entre algunas otras tecnologías donde teníamos más opciones. En primer lugar, debíamos elegir un Framework y un entorno de desarrollo para la implementación de nuestra aplicación móvil. Nos hemos decidido por utilizar AndroidStudio, descartando otras dos opciones que habíamos valorado.

Por un lado, valoramos la opción de desarrollar la aplicación en Xamarin. Para programar en este Framework se utiliza C#. Ofrece algunas facilidades para desarrollar aplicaciones multiplataforma que no están disponibles en AndroidStudio. Aunque la programación es muy similar a la que se pueda realizar en AndroidStudio el hecho de que el lenguaje sea C# nos podría ocasionar algún problema con la librería OpenCV. Además, para el desarrollo de aplicaciones multiplataforma Xamarin ofrece muchas facilidades. Se pueden reutilizar gran parte de nuestro código entre una aplicación para Android e iOS sin tener que portar desde Java a Objective-C y viceversa.

Por otro lado, se valoró la opción de utilizar Apache Cordova. El lenguaje utilizado sería en este caso lenguaje web: HTML, CSS y JavaScript. En este Framework es posible desarrollar una aplicación multiplataforma sin prácticamente hacer trabajo adicional. Las limitaciones que nos supone el uso de lenguaje web para el desarrollo de una interfaz gráfica potente nos hizo descartar esta opción. Además, esta última opción, mucho menos conocida que las anteriores, supondría un riesgo adicional ya que en la web es más difícil encontrar información que nos pudiera resultar útil.

Hemos consultado información en la web y algunos ejemplos de aplicaciones desarrollados en cada uno de estos Framework. Tras este trabajo de investigación, hemos llegado a la conclusión de que muchas de las empresas que quieren realizar una aplicación multiplataforma lo que deciden es desarrollar parte de la aplicación paralelamente en las diferentes plataformas. Lo que nos parece más interesante es

utilizar una lógica de negocio común y desarrollar una interfaz gráfica personalizada para cada plataforma, aunque parte de esta interfaz sea compartida.

Estos motivos nos han hecho descartar las dos anteriores opciones. Hemos descartado Apache Cordova por las limitaciones que nos suponía el desarrollo de interfaces haciendo uso de lenguajes web. En Xamarin sería necesario hacer un gran trabajo adicional para lograr que nuestra aplicación funcionase también en iOS o en otra plataforma. Esto escapa de las posibilidades iniciales de este trabajo. Por tanto, como vamos a desarrollar una aplicación móvil para el sistema operativo Android, hemos escogido AndroidStudio. Programar en Java será sin duda más adecuado para este trabajo que hacerlo en C#.

Metodología de trabajo

Vamos a seguir una metodología de desarrollo ágil con un proceso de desarrollo iterativo-incremental. Así, dividiremos las 300 horas de trabajo en 15 sprints de 20 horas que abarcarán una semana cada uno. En cada uno de estos sprints se realizarán tareas variadas. En los primeros, como es normal, se dedicará más tiempo al diseño pero también será necesario implementar algo. A medida que avancemos con el trabajo, aunque haya que diseñar algo más, el peso de la implementación será más importante que en anteriores sprints. Hacia el final del trabajo, tendremos que dedicar más tiempo a probar la aplicación y a corregir errores. Durante todo el proceso documentaremos al mismo tiempo que desarrollamos.

Para completar los 15 sprints trabajaremos 10 semanas consecutivas, comenzando el 30 de enero y parando 9 de abril. La semana del 10 al 16 de abril descansaremos, retomando a la semana siguiente el trabajo. Por tanto, salvo que surja algún imprevisto, se espera que la aplicación esté desarrollada el día 19 de mayo. Si surge algún problema que nos obligue a retrasar la entrega tendremos todavía margen suficiente para solucionarlo.

Planificación

Con este apartado se pretende marcar la hoja de ruta que seguirá el trabajo. Aunque estará sujeta a posibles cambios, esta planificación será el mejor indicador de cómo avanza el trabajo. A continuación, se propone para cada uno de los 15 sprints en los que se divide el trabajo una serie de tareas. Es probable que en alguno de los sprints no se terminen todas las tareas. Estas tareas sin terminar se retomarán en el sprint siguiente. Por otro lado, si en alguno de ellos se terminan las tareas antes de que se agoten las 20 horas se cogerán tareas del siguiente. Aunque en la lista de tareas no hay horas explícitas, se entiende que el conjunto de tareas de cada sprint suman un total de 20 horas de dedicación.

- Sprint 1: Desarrollar la introducción del trabajo. Estudiar la aplicación de escritorio AntibiogramJ y las librerías OpenCV e ImageJ. Definir los objetivos del trabajo y detallar la planificación.

- Sprint 2: Diseñar la interfaz de la aplicación móvil para pantallas grandes. En un principio, se prevé que la aplicación se utilice con el dispositivo orientado en horizontal, por lo que se pensará en diseños de interfaz de este tipo. Empezar a implementar las diferentes pantallas de la interfaz.

- Sprint 3: Terminar con la implementación de la interfaz para dispositivos móviles de pantalla grande orientada en sentido horizontal.

- Sprint 4: Estudiar el sistema gestor de base de datos utilizado en AntibiogramJ. Estudiar la tecnología que nos permita la posterior creación de la base de datos. Crear la base de datos sobre el dispositivo móvil.

- Sprint 5: Hacer el diseño de la capa de persistencia. Esta se utilizará para manejar los datos relativos a bacterias, antibióticos, usuarios, etc. Implementar la capa de persistencia.

- Sprint 6: Analizar y diseñar la lógica de negocio que subyacerá bajo nuestra aplicación. Como parte final de este sprint se pretende obtener una lista detallada de requisitos funcionales que reflejen el funcionamiento del sistema. Implementar la carga de imágenes en la aplicación.

- Sprint 7: Gestionar la creación de experimentos y antibiogramas. Completar la implementación de las funciones que permitan realizar la carga de imágenes en la aplicación desde la galería y/o tomándolas con la cámara de fotos. Continuación del estudio de OpenCV e ImageJ y conexión con AndroidStudio.

- Sprint 8: Haciendo uso de las librerías OpenCV e ImageJ implementar las funciones que permitan realizar el paso 1 y paso 2 para el estudio de un antibiograma.

- Sprint 9: De nuevo, haciendo uso de las librerías OpenCv e ImageJ, implementar las funciones que permitan realizar los pasos 3 y 4 para el estudio de un antibiograma.

- Sprint 10: Implementar las funciones que permitan realizar el paso 5 para el estudio de un antibiograma. Gestionar el guardado y la carga de experimentos.

- Sprint 11: Estudio de los algoritmos utilizados para la detección automática en las imágenes. Implementación de mejoras sobre los algoritmos de detección automática ya implementados.

- Sprint 12: Realizar un testeo global de la aplicación. Preparar la aplicación para superar una prueba de aceptación con el cliente. Corregir los posibles errores que se hayan detectado. Redactar la conclusión del trabajo.

- Sprint 13: Realizar el diseño de la interfaz para dispositivos móviles Android con pantallas pequeñas. Implementar esta interfaz.

- Sprint 14: Realizar los ajustes necesarios para que la interfaz de dispositivos móviles Android con pantalla pequeña funcione correctamente.

- Sprint 15: Terminar con la interfaz para dispositivos de pantalla pequeña. Conectar la lógica de negocio, dotando de la funcionalidad necesaria a la interfaz para dispositivos de pantalla pequeña.

Al comienzo de cada sprint detallaremos las tareas a realizar y al final del mismo haremos un pequeño análisis de las tareas realizadas y las que quedarán pendientes para futuros sprints. De esta manera, si es necesario replanificar, se hará antes de comenzar un nuevo sprint.

Desarrollo de la aplicación

En este apartado de la memoria iremos explicando el proceso de desarrollo de la aplicación y siguiendo su evolución. Está compuesto por los 15 sprints en los que desarrollaremos el trabajo. Para cada sprint habrá tres subapartados en los que se explicarán respectivamente las tareas previstas, el desarrollo del sprint y el grado de cumplimiento de las tareas planificadas al comienzo del mismo. En el subapartado dedicado al desarrollo del sprint se incluirán los problemas que vayan surgiendo y la manera de resolverlos. Además, incluiremos los conocimientos más importantes adquiridos durante las 20 horas de trabajo.

Sprint 1

Tareas a realizar en este sprint.

- Estudiar la aplicación de escritorio AntibiogramJ.
- Estudiar superficialmente las librerías OpenCV e ImageJ.
- Realizar la introducción del trabajo.
- Definir los objetivos del trabajo y elaborar una planificación.

Desarrollo del sprint.

Para realizar una versión móvil de la aplicación AntibiogramJ, en primer lugar, hemos estudiado su funcionamiento. Para ello, además de poder ejecutarla y realizar varios antibiogramas hemos tenido acceso al manual de la aplicación. En éste se explica no solo el funcionamiento básico de la aplicación, si no también otras características que no se pueden apreciar a simple vista. Con este manual hemos podido conocer cómo la aplicación accede a las tablas EUCAST. En este primer sprint, además de estudiar la aplicación de escritorio, hemos investigado sobre las tecnologías que tendremos que utilizar. Las tecnologías ImageJ y OpenCV, mencionadas anteriormente, eran hasta el momento desconocidas para mí.

Como hasta ahora nunca antes había trabajado con AndroidStudio, hemos dedicado también algo de tiempo a conocer este entorno de desarrollo. En lugar de estudiarlo profundamente desde un principio, será más productivo ir aprendiendo a usar funcionalidades nuevas a medida que las vayamos necesitando. Por tanto, durante todo el trabajo iremos aprendiendo sobre AndroidStudio y sobre cualquier otra herramienta que podamos utilizar más adelante.

En estas primeras 20 horas de trabajo también hemos planificado el total desarrollo de este trabajo fin de grado. Si bien es cierto que podrá sufrir alguna pequeña desviación, nos será de gran ayuda para saber en cada momento cuál es el siguiente paso a dar. También nos permitirá tener una excelente guía para ver si el ritmo al que avanzamos es el adecuado.

Como resultado final de esta fase de trabajo, hemos redactado la introducción de este TFG. En ella se incluyen entre otros, como ya habrá podido observar el lector, los objetivos, la planificación y las tecnologías empleadas.

Resultados alcanzados en el sprint

En este primer sprint hemos cumplido con todas las tareas que se habían propuesto al comienzo del mismo.

Sprint 2

Tareas a realizar en este sprint.

- Diseño de la interfaz móvil para pantallas grandes.
- Estudio de buenas prácticas para implementar interfaces en Android.
- Implementación de las interfaces de los pasos 1, 2 y 3.

Desarrollo del sprint.

Ya que el tiempo del que disponemos para realizar este trabajo es limitado hemos optado por hacer, sobre papel, un diseño de la interfaz de usuario que la aplicación tendrá. Como ya disponíamos de la versión de escritorio, un pequeño boceto ha sido suficiente y no hemos utilizado ninguna herramienta software de diseño ya que nos hubiera llevado más tiempo. Además, el disponer de una versión de escritorio ya implementada nos ha permitido tener claro el funcionamiento de la aplicación desde un principio, facilitando enormemente la fase de diseño de la interfaz. En las Figuras 6 y 7 se muestran algunas de las pantallas que hemos dibujado en esta fase. Todas ellas son relativas al proceso de realización de un experimento: pasos 1 y 2.1 respectivamente. Recordemos que estos pasos son los que aparecen reflejados en el apartado de antecedentes de esta memoria.

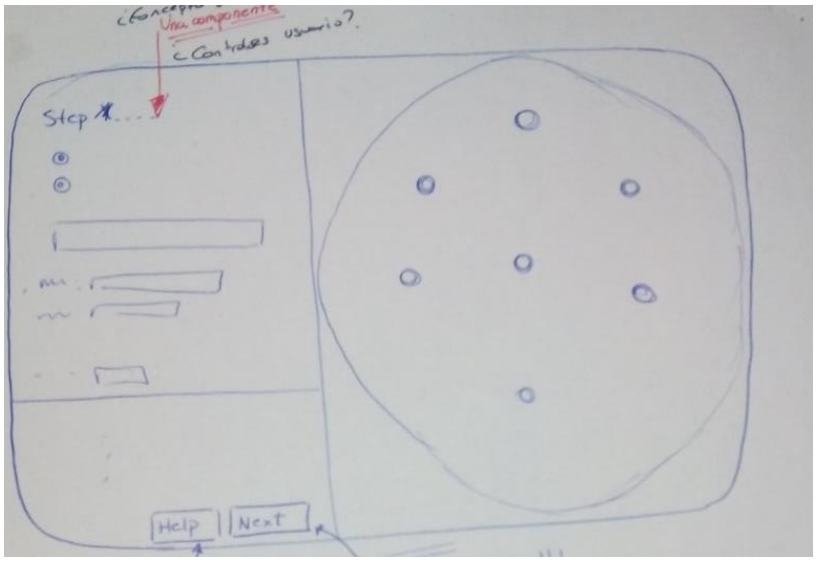

Figura 6: Diseño de la interfaz sobre papel. Paso 1

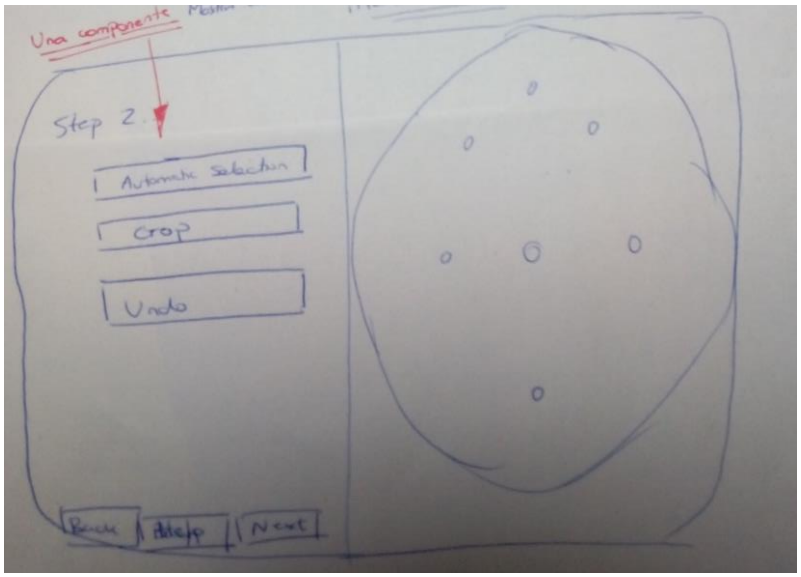

Figura 7: Diseño de la interfaz sobre papel. Paso 2.1

Tras obtener el visto bueno por parte del cliente, y antes de empezar con la implementación de las interfaces, hemos observado que las pantallas relativas a la realización del experimento comparten un esquema muy similar. Los diferentes controles que el usuario puede manipular se sitúan a la izquierda, mientras que ocupando el resto de la pantalla habrá una imagen. Además, en la parte de los controles habrá tres botones que aparecerán en todas las pantallas: Back, Help y Next como podemos ver en las Figuras 6 y 7.

Tras investigar las distintas posibilidades que teníamos para implementar las pantallas que permiten realizar cada experimento hemos encontrado algo que se ajusta muy bien a nuestras necesidades: los Fragments. De acuerdo con la Android Developer Guide [2], un Fragment representa un comportamiento o una parte de la interfaz de usuario en una actividad o Activity. En una misma actividad pueden usarse varios Fragments, y cada uno de ellos puede ser utilizado en varias actividades.

En nuestro caso, los utilizaremos para construir los controles que el usuario puede manipular en cada paso de un experimento. Así para todos los pasos de la realización de un experimento se comparte un mismo Layout, y lo único que cambia son los controles de la izquierda. Además, en una posible futura implementación de una versión de la aplicación para dispositivos con un tamaño de pantalla menor podremos reutilizar los fragmentos que ya habremos implementado.

Otra de las grandes ventajas que usar fragmentos nos ofrece es que manejarlos es bastante sencillo. Android dispone de una clase llamada `FragmentManager` que permite añadir o quitar fragmentos de un Activity dinámicamente.

Además de diseñar la interfaz de la aplicación y pensar en su funcionamiento, hemos empezado con su implementación. Fijándonos en el diseño de la interfaz, hemos empezado por implementar los controles que aparecerán en cada paso durante la realización de un experimento. Todos tienen un LinearLayout y extienden de la clase

`Fragment`. En la Figura 8 se puede ver el aspecto del control del paso 1. En la Figura 9 podemos ver la implementación del control del paso 2.1.

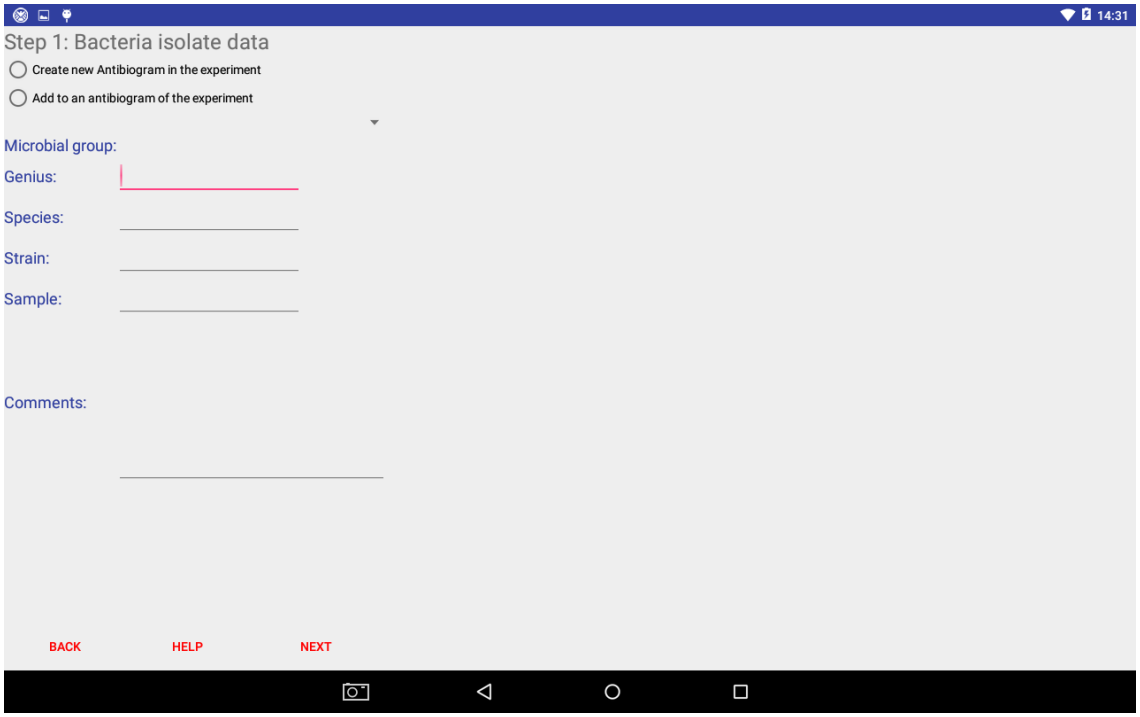

Figura 8: Captura de pantalla de la realización de un experimento. Paso 1

Figura 9: Captura de pantalla de la realización de un experimento. Paso 2.1

Cuando ya teníamos todos los controles implementados, en total ocho, haciendo uso de `FragmentManager` hemos implementado la funcionalidad del botón Next de forma

que cambie el control cuando se pulse. De forma similar a como hemos hecho para el botón Next, programaremos el cambio de controles cuando el botón pulsado sea Back.

Resultados alcanzados en el sprint

En este sprint casi se han completado en su totalidad las tareas planificadas. Falta por implementar la interfaz del paso 3.3 que quedará como tarea pendiente para el próximo sprint.

Sprint 3

Tareas a realizar en este sprint.

- Implementación de las interfaces de los pasos 3.3 , 4 y 5
- Diseño e implementación de las pantallas iniciales de la aplicación.

Desarrollo del sprint.

En este sprint continuaremos con las labores de implementación de la interfaz que estábamos haciendo en el anterior. En primer lugar, hemos añadido funcionalidad al botón Back, que simplemente cambia el Fragment con los botones que aparecen en cada momento.

Además hemos implementado los controles de los pasos 3.3, 4 y 5. En este apartado nos han surgido muchos problemas para implementar el control del paso 3.3. Como podemos ver en la Figura 10, para cada disco de antibiótico que aparezca en la imagen habrá en el control un botón y una lista desplegable que permita elegir cuál es.

Figura 10: Captura de pantalla de la realización de un experimento. Paso 3.3

Dado que en cada imagen el número de discos de antibióticos cambiará, no podemos saber de antemano cuántos hay. Así la generación de este control se produce necesariamente en tiempo de ejecución, es decir, una vez que el programa conoce el número de discos de antibiótico que hay. Nos ha llevado bastante tiempo conseguir que el comportamiento de este control fuese satisfactorio. Tras unas cuantas horas dedicadas a leer información en la Android Developer Guide y múltiples intentos, hemos conseguido que funcionara.

En este mismo sprint también hemos diseñado e implementado la pantalla principal de la aplicación. Con respecto a la aplicación de escritorio, esta es la pantalla que más cambia. Cuando un usuario maneja una aplicación para un ordenador está acostumbrado a hacer clic sobre menús que se despliegan y que muchas veces contienen a su vez múltiples submenús. Esto no es una práctica habitual en las aplicaciones móviles. Para la versión móvil de esta aplicación hemos optado por cambiar el diseño de esta pantalla tal y como se muestra en la Figura 11.

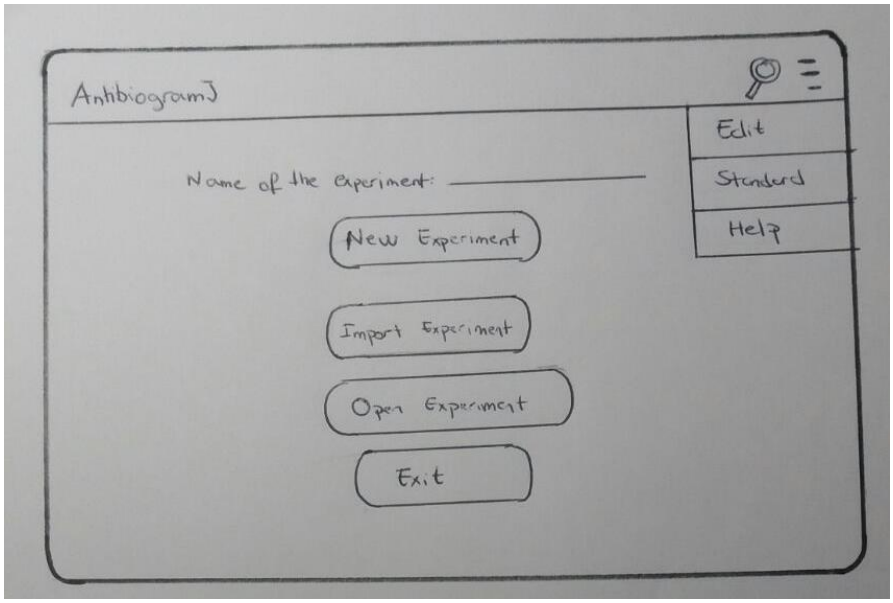

Figura 11: Diseño de la interfaz sobre papel. Pantalla inicial

Como vemos en esta Figura, aparecen 4 botones. De arriba a abajo permitirán respectivamente: crear un experimento, importar un experimento, abrir un experimento y cerrar la aplicación. En la aplicación de escritorio estas 4 funcionalidades eran opciones del elemento “File” del menú.

En la parte superior de la pantalla aparece un menú usual en aplicaciones móviles. Dispone de dos botones que permitirán hacer búsquedas y desplegar las opciones. Las opciones son: “Edit” para configurar las abreviaturas de los antibióticos; “Standar” para poder modificar el estándar usado y “Help” que redirige al usuario a la página donde se detalla la información de la aplicación.

Como comentamos anteriormente, un experimento puede estar compuesto por varios antibiogramas. Estos a su vez pueden tener varias imágenes. Por tanto, tendremos una segunda pantalla para el experimento creado en la que se podrá: cargar una imagen, exportar un antibiograma o exportar un experimento. De esto modo estaremos

ofreciendo la misma funcionalidad que ofrece la versión de escritorio. En la Figura 12 se muestra el diseño de esta segunda pantalla.

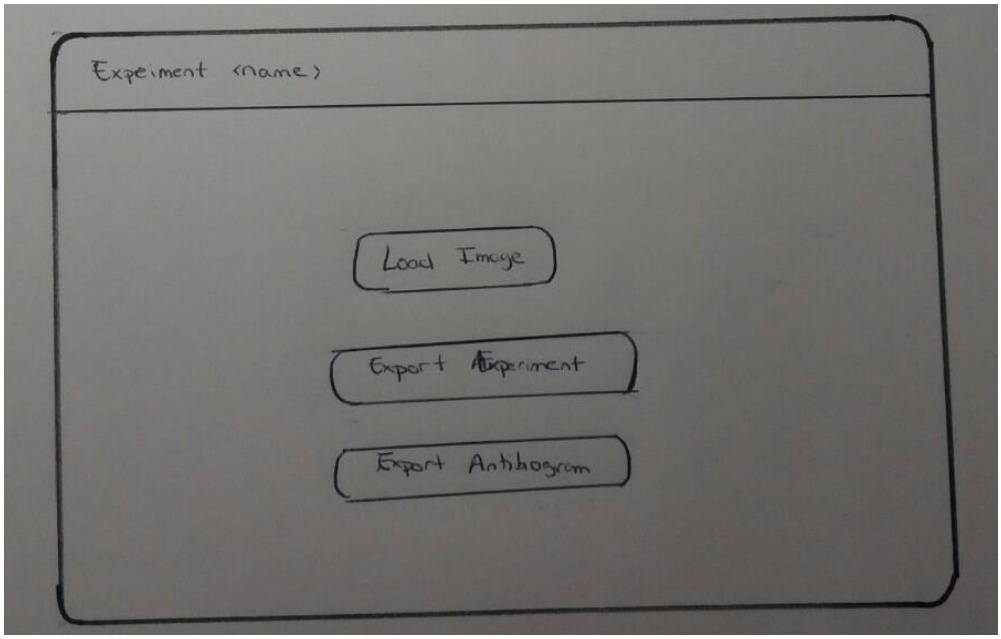

Figura 12: Diseño de la interfaz sobre papel. Experiment <name>

Una vez que tenemos claro el diseño de estas pantallas, se las enseñamos a los clientes. Cuando nos han dado su visto bueno nos ponemos a implementarlas. Lo que más tiempo nos ha llevado ha sido implementar el menú superior de la pantalla inicial. Aunque finalmente no ha resultado ser muy complicado hemos dedicado mucho tiempo a buscar información acerca de cómo hacerlo. En las Figuras 13 y 14 se puede ver el aspecto que tienen estas interfaces cuando las ejecutamos.

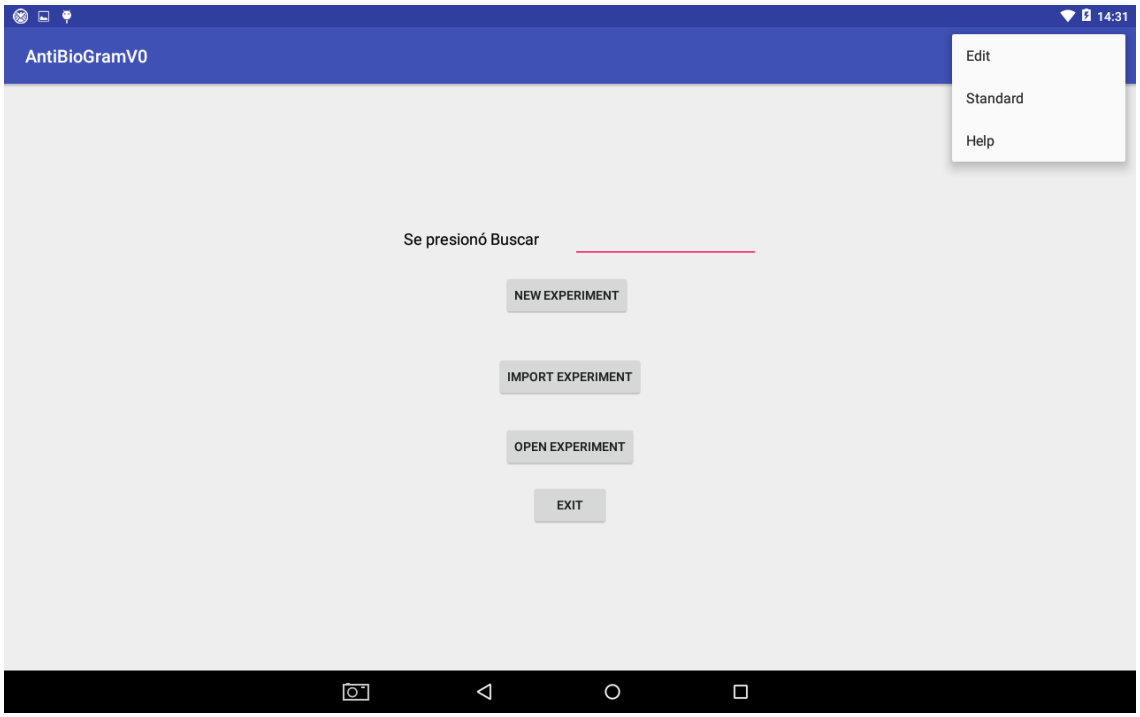

Figura 13: Interfaz de la aplicación. Pantalla inicial

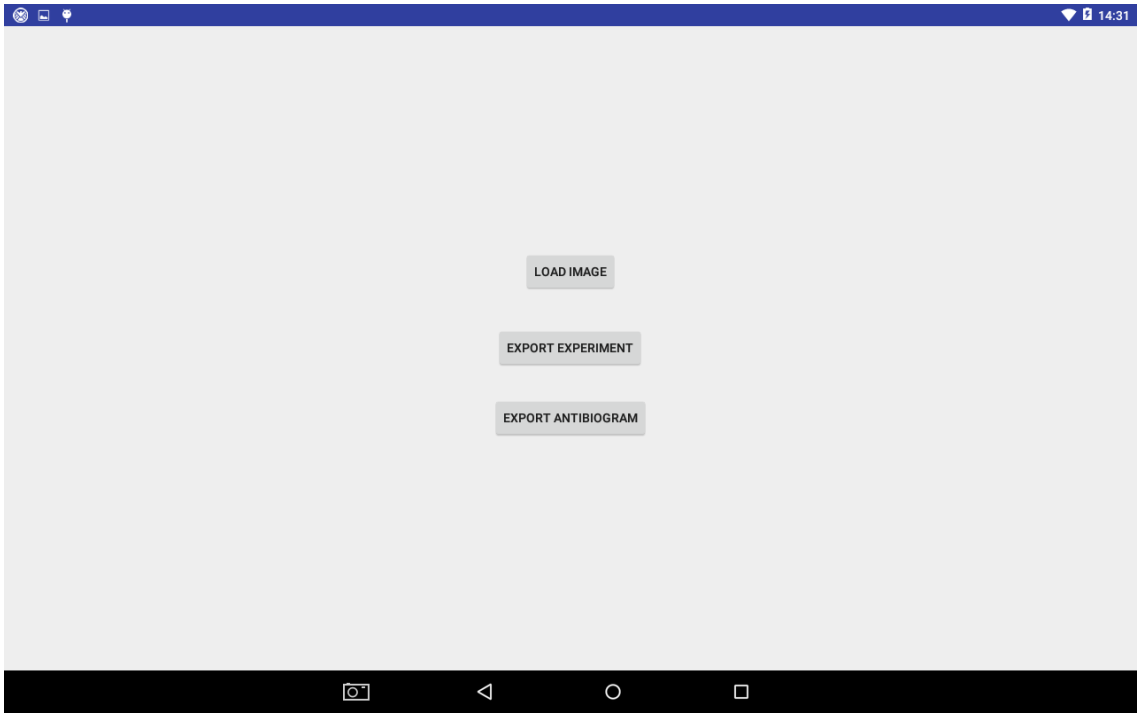

Figura 14: Interfaz de la aplicación. Experiment <name>

Hasta aquí ha llegado el desarrollo de este sprint. Como resultado de los sprints 2 y 3 tenemos implementada una interfaz para la aplicación móvil. Queda pendiente una nueva reunión con el cliente para obtener su aprobación final sobre toda la interfaz. Dado que la semana que viene se encontrará de viaje, seguiremos con otras tareas, posponiendo así la reunión hasta la semana siguiente.

Resultados alcanzados en el sprint

En este sprint hemos cumplido con todas las tareas que se habían propuesto al comienzo del mismo.

Sprint 4

Tareas a realizar en este sprint.

- Estudio de las diferentes opciones y selección de un sistema gestor de base de datos (SGBD) para la aplicación.
- Creación de la base de datos en Android.
- Implementación de la capa de persistencia de la aplicación.

Desarrollo del sprint.

En este sprint vamos a trabajar en la capa de persistencia de la aplicación. Para la aplicación de escritorio ya se dispone de un SGDB implementado haciendo uso de MySQL. El esquema de la base de datos es el que se muestra en la Figura 15.

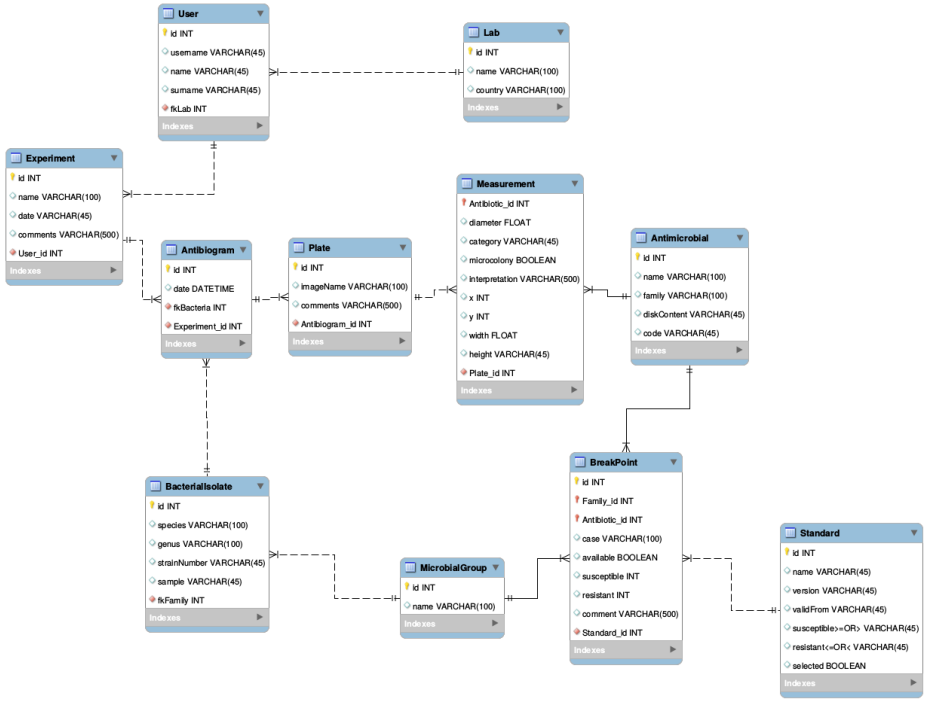

Figura 15: Esquema de la base de datos de AntibiogramJ.

En nuestra aplicación móvil no incluiremos en un principio toda esta información. Dado que la capacidad de los dispositivos móviles es más limitada, hemos optado por quitar, de momento, todo lo referido a usuarios y laboratorios.

El motor de bases de datos que vamos a utilizar en Android es SQLite. Según su web oficial [4], las ventajas que nos ofrece son: su transaccionalidad, su tamaño reducido, el que no necesite servidor y el que precise de poca configuración. Además Android incorpora una API para la creación y gestión de este tipo de bases de datos, denominada SQLiteDatabase.

Con ayuda de la aplicación SQLiteStudio y, siguiendo su manual, hemos creado la base de datos. Para llegar a esto hemos tenido multitud de problemas. Al ser una tecnología totalmente desconocida para mí, ha sido necesario dedicarle algo de tiempo a su estudio.

El primer intento de creación de la base de datos lo hemos hecho con la línea de comandos. No parecía muy complicado y, para una base de datos tan sencilla como esta, pensamos que sería suficiente. Sin embargo, requirió de muchos intentos, y dedicamos demasiado tiempo en algo que acabó por no servir para mucho.

A continuación, investigamos en internet y descubrimos que existía la opción de crear bases de datos SQLite con una interfaz gráfica: SQLiteStudio. Tras un arduo trabajo con intentos fallidos conseguimos crear la base de datos.

Para cuando nos quisimos dar cuenta ya se había terminado el sprint, y lo único que teníamos era una base de datos SQLite vacía en la que había que insertar muchos datos. Además esta base de datos estaba hecha en Windows y hacerla en Android supondría un trabajo adicional.

Resultados alcanzados en el sprint

Debido al desconocimiento de la tecnología a utilizar en este sprint han surgido problemas y queda pendiente la creación de la base de datos y la implementación de la capa de persistencia.

Sprint 5

Tareas a realizar en este sprint.

- Creación de la base de datos.
- Implementación de la capa de persistencia de la aplicación.
- Reunión con el cliente para la aprobación de la interfaz implementada.

Desarrollo del sprint.

Al comienzo de esta semana hemos mantenido una reunión con el cliente para presentarle el primer prototipo de la interfaz de la aplicación. En general está satisfecho con el trabajo realizado pero hay que hacer algunas modificaciones. Con las interfaces propuestas no se muestra claramente el hecho de que un experimento esté formado por varios antibiogramas. Además cada uno de ellos puede estar formado por varias imágenes. Tras anotar las indicaciones del cliente hemos comenzado el trabajo de este sprint reimplementando la interfaz de la aplicación. No nos ha supuesto mucho tiempo, así que podremos proseguir con el trabajo planificado para el sprint.

Los nuevos cambios en la interfaz afectan a las Figuras 13 y 14 mostradas anteriormente. En las Figuras 16 y 17 podemos ver cómo han quedado ahora estas pantallas. En la primera de ellas vemos que ya no se pide el nombre del experimento, pues solicitarlo tiene sentido una vez que ya se ha pulsado el botón “New Experiment”.

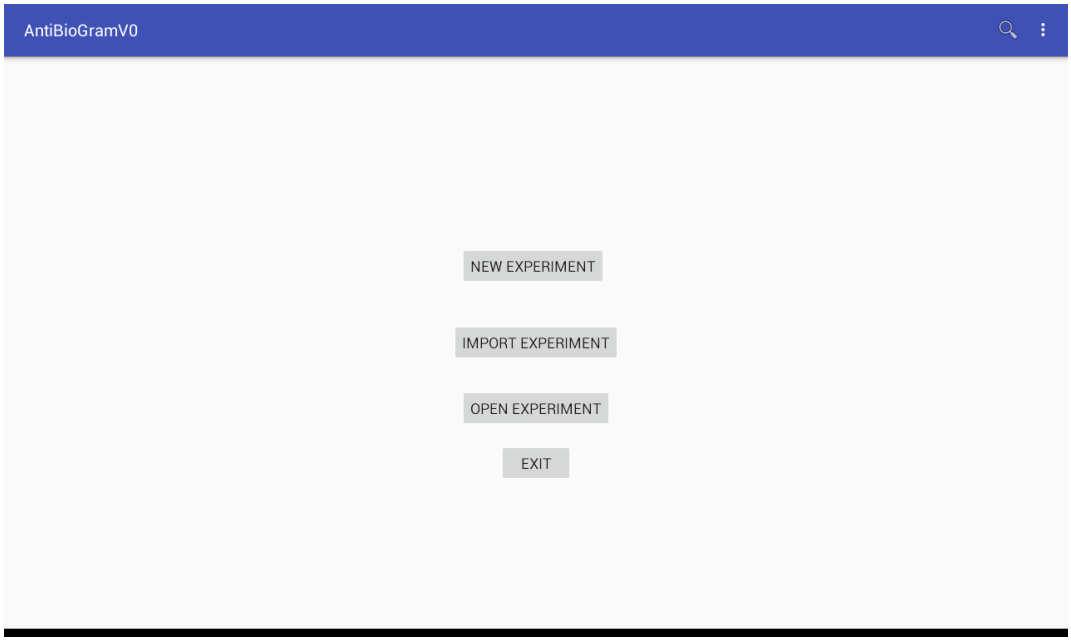

Figura 16: Nueva interfaz de la aplicación. Inicio

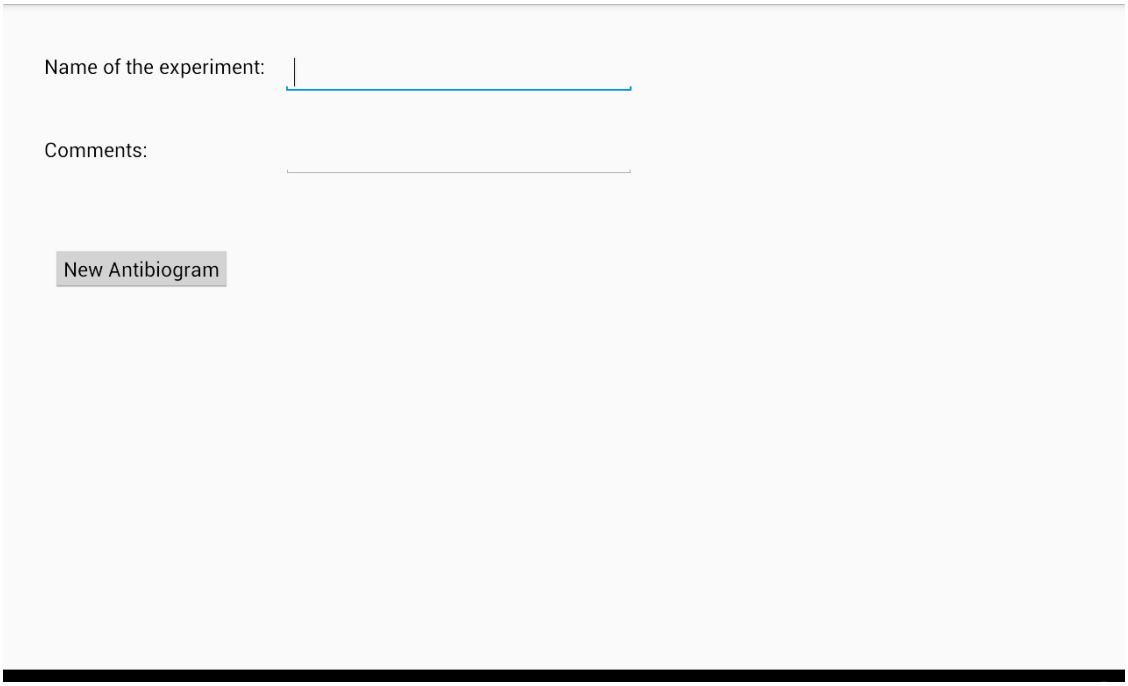

Figura 17: Nueva interfaz de la aplicación. Nuevo experimento

En la Figura 16 vemos que ahora lo único que se ofrece es la opción de proporcionar los datos del experimento y la de crear un nuevo antibiograma. Cuando el usuario hace clic en el botón “New Antibiogram”, la aplicación le lleva a la pantalla que se muestra en la Figura 18. En ésta, se debe introducir los datos referidos al antibiograma y se ofrece la opción de añadir una nueva imagen haciendo clic en el botón “Analize Image”.

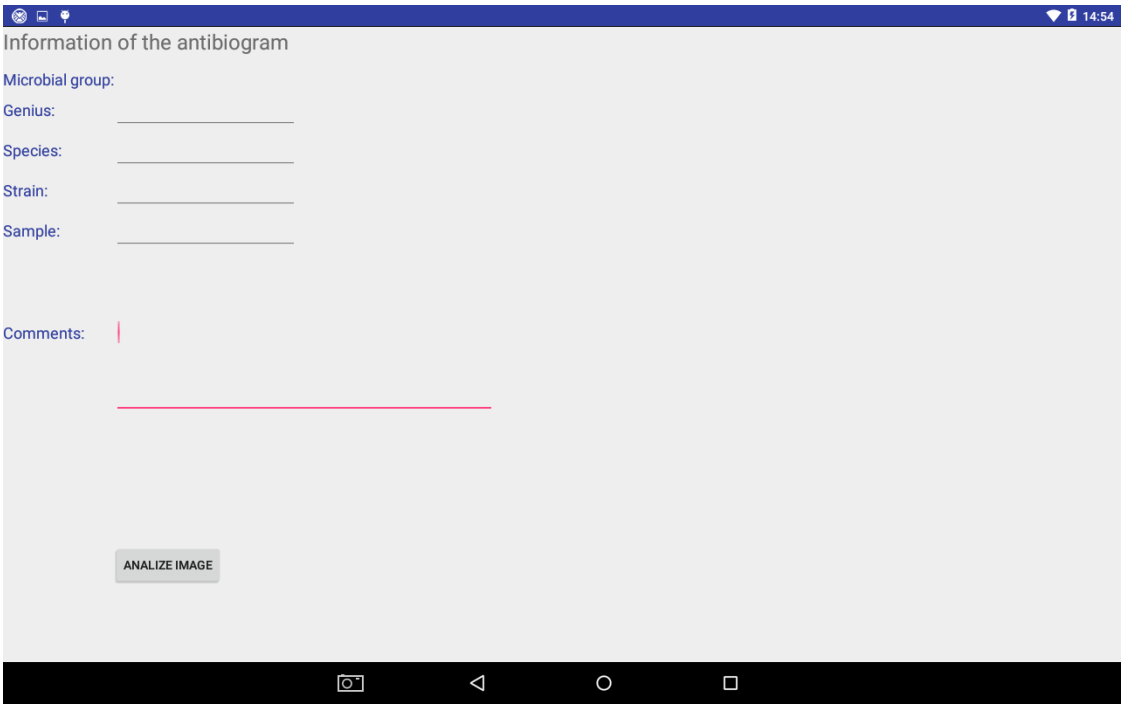

Figura 18: Nueva interfaz de la aplicación. Nuevo antibiograma

Como consecuencia del retraso sufrido en el anterior sprint, durante esta semana, continuaremos con las tareas de la persistencia de la aplicación pese a que en la planificación se indicia que íbamos a empezar con la lógica de negocio.

En primer lugar, nos gustaría mencionar el canal de YouTube Código Alonso [5]. En un principio no sabíamos nada de bases de datos SQLite y ha sido necesario buscar y buscar información por la red para poder aprender. Aunque muchas de las webs que hemos leído han sido útiles, este canal de YouTube ha sido la mejor forma de aprender a crear una base de datos, insertar datos en ella y recuperarlos posteriormente para su uso. Son los tutoriales 14, 15 y 16 de este canal.

Durante la primera parte de trabajo de esta semana, hemos estado creando la base de datos en el móvil. Para ello, creamos las tablas desde una clase que hemos llamado `DbHelper`, que extiende de `SQLiteOpenHelper` y cuya función es la creación y gestión de la base de datos. Desde el Android Device Manager debiéramos ser capaces de poder ver el fichero en el que se ha creado la base de datos dentro del dispositivo móvil. El problema es que para poder ver este fichero es necesario tener el dispositivo rooteado.

Como el dispositivo sobre el que estábamos probando la aplicación no era nuestro no podíamos rootearlo. Para comprobar que la base de datos funcionaba tuvimos que trabajar algo más e ir mostrando una serie de mensajes por pantalla.

Una base de datos SQLite está en un solo archivo, a diferencia de lo que ocurre en MySQL, por ejemplo. Un atributo que aparecerá en la clase `DbHelper` es `DB_SCHEME_VERSION`, que se cambia cuando modificamos el esquema de la base de datos. De esto se encarga el método `onUpgrade`. A continuación, explicamos para qué hemos utilizado esta clase.

En este sprint están empezando a cambiar las cosas. Hemos conseguido crear la base de datos SQLite. Para ello, hemos hecho uso de un par de clases adicionales: `DbHelper` y `DbManager`. Con estas dos clases manejamos la creación de la base de datos. En la clase `DbManager` tenemos definido el esquema y la clase `DbHelper` es la que se encarga de crear las tablas haciendo uso de las sentencias que se encuentran en el `DbManager`. Luego, el `DbManager` le pide al `DbHelper` la base de datos.

Una vez que la base de datos ya está creada lo que hacemos es insertar unos cuantos datos. En sprints posteriores se pretende poblar la base de datos con los datos de las tablas EUCAST. Una vez que algunos datos ya están insertados, haciendo uso de cursores, hemos hecho algunas pruebas accediendo a algunos de ellos. Con el depurador de AndroidStudio se puede ver el contenido de la base de datos, pero para ello es necesario que el dispositivo móvil esté rooteado. Como nuestro dispositivo no lo está, optamos por realizar algunas consultas para comprobar que la base de datos funciona.

Ahora que la base de datos ya funciona, lo que hacemos es crear una sola instancia de la clase `DbManager` que sea accesible desde toda la aplicación. Para ello, implementamos un método, que denominamos `getManager`, y que de forma similar

al patrón Singleton genera una única instancia de esta clase. En esta clase hemos implementado algunos métodos de acceso a la base de datos. A medida que nos hagan falta otros métodos de acceso o modificación de la base de datos, los iremos implementando.

Resultados alcanzados en el sprint

En este sprint hemos conseguido alcanzar los objetivos propuestos. Queda pendiente para un futuro la población completa de la base de datos. Para esto último haríamos uso además de las tablas EUCAST. De momento consideramos que esta labor no es prioritaria ya que con la inserción de unas cuantas filas ya seremos capaces de probar la aplicación.

Sprint 6

Tareas a realizar en este sprint.

- Captura más detallada de los requisitos funcionales de la aplicación.
- Implementación de la carga de imágenes desde la galería.
- Implementación de la toma de imágenes con la cámara de fotos.

Desarrollo del sprint.

Una vez que ya tenemos implementada la interfaz de la aplicación y la capa de persistencia, ha llegado el momento de empezar con la lógica de negocio de la aplicación. Como primera tarea de este sexto sprint, realizamos una captura de los requisitos funcionales que la lógica de negocio debe satisfacer. Estos son los que listamos a continuación:

- Carga de imágenes desde la galería.
- Carga de imágenes tomándolas directamente con la cámara de fotos.
- Detección automática del borde de una placa de Petri.
- Selección manual del borde de una placa de Petri.
- Recorte de una imagen tras la detección/selección del borde de una placa de Petri. Deshacer esta operación.
- Ajuste manual y automático del brillo y del contraste de la imagen.
- Detección automática de los discos de los antibióticos que se colocan en la placa de Petri.
- Selección manual de estos discos.
- Detección automática del diámetro de un disco de antibiótico y selección de su medida manualmente.
- Selección manual del diámetro de un disco de antibiótico.
- Lectura automática del nombre del antibiótico que hay en cada disco.
- Selección manual del nombre del antibiótico que hay en cada disco.
- Selección de los valores que indican la zona de inhibición y la zona de susceptibilidad.
- Visualización de la zona de inhibición y la zona de susceptibilidad.

- Anotación de la información referida a la zona de inhibición y a la zona de susceptibilidad. Anotación de otra posible información generada durante la realización del experimento.

Como puede observar el lector, los requisitos funcionales que aquí presentamos son de grano grueso. Mientras estemos implementando los métodos que permitan a nuestra aplicación satisfacer los requisitos funcionales, iremos documentándolo en esta memoria. De esta forma cada requisito quedará desgranado.

Los dos primeros requisitos funcionales que nuestra lógica debe satisfacer son la carga de imágenes desde la galería y la captura de una foto con la cámara del dispositivo móvil. Para ello, lo que vamos a hacer es generar dinámicamente un menú en el que se ofrece al usuario dos opciones: seleccionar la imagen desde la galería o tomarla con la cámara de fotos. Para generar este menú hacemos uso de `AlertDialog`, cuya especificación completa se puede encontrar en la Android Developer Guide.

Cuando el usuario pulsa en el botón “Analize Image”, que se puede ver en la Figura 18, la aplicación se dirige a la pantalla en la que se cargará la imagen. Antes de nada, despliega un diálogo, que ofrece al usuario dos opciones. Por un lado, puede elegir cargar una imagen desde la galería mediante la opción “Choose from Library” y , por otro lado, puede decidir tomar una foto con la cámara con la opción “Take Photo”. La foto se situará sobre el ImageView, que ocupa casi toda la pantalla y se sitúa a la derecha de los controles. En la Figura 19 podemos ver una captura de este paso.

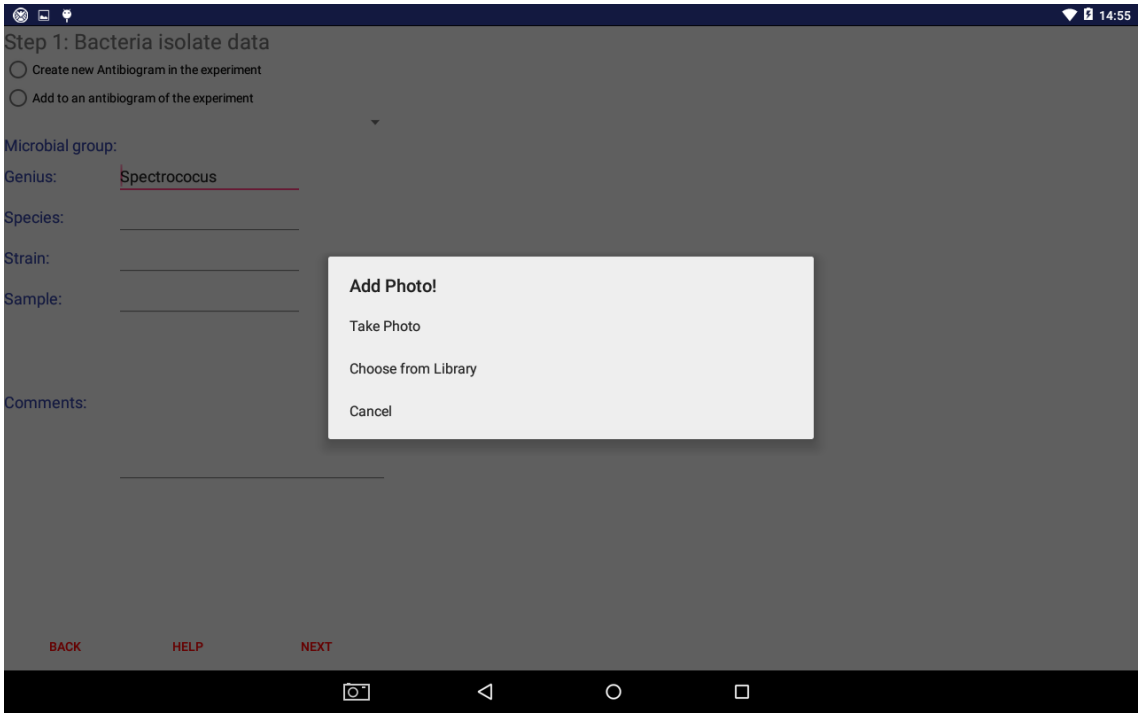

Figura 19: Diálogo para elegir la opción para cargar una foto

Sea cual sea la elección del usuario, lo que hacemos es generar una Intent. De acuerdo con la Android Developer Guide, una Intent es un objeto de acción que se puede usar para solicitar una acción de otro componente de la aplicación. Así lo que haremos será solicitar a la aplicación que, o bien abra la cámara para tomar una fotografía, o bien abra la galería para seleccionar una de las imágenes guardadas. Una vez que hemos creado el Intent correspondiente de acuerdo a la elección del usuario, la aplicación llama a un método u otro.

Veamos en primer lugar cómo funciona el método que permite cargar la imagen desde la galería. Este método hace uso de la función `getBitMap` de la clase `MediaStore`. Uno de los parámetros que esta función recibe es la ubicación del archivo. Lo que hace el método es poner en el mapa de bits del `ImageView` la imagen seleccionada. Para hacer esto hacemos uso del método `setBitMap` de la clase `ImageView`. En la Figura 20 se puede ver una captura de pantalla de esta parte del proceso.

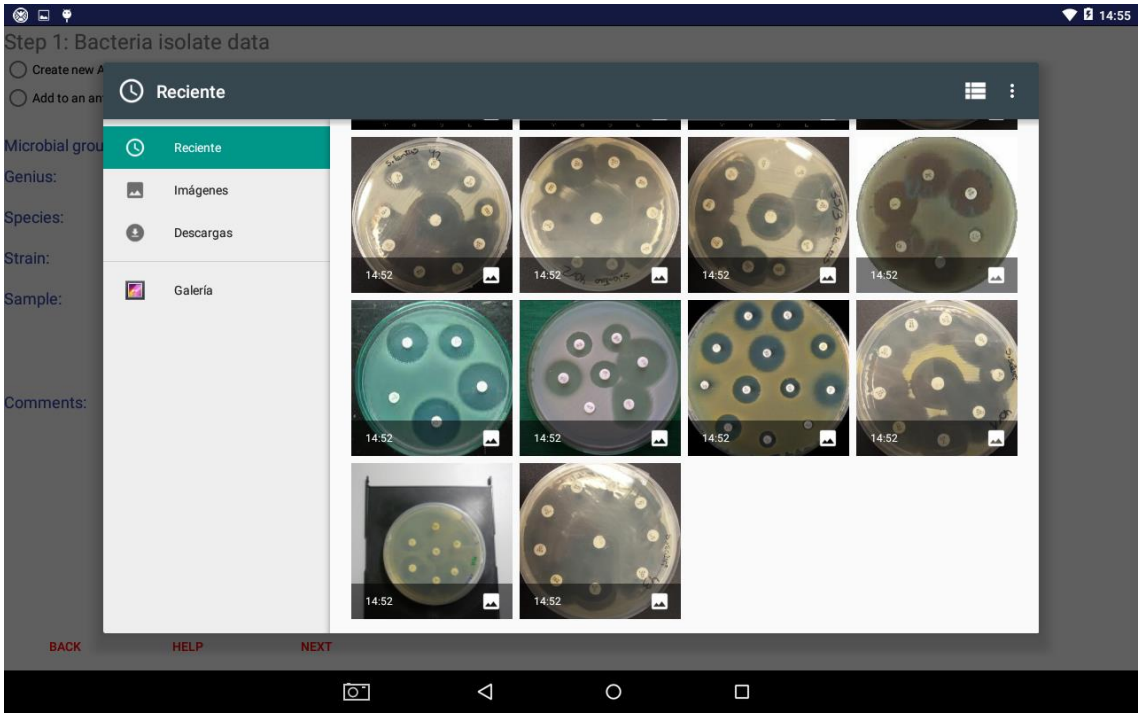

Figura 20: Selección de la imagen desde la galería

Para la captura de la fotografía usando la cámara el proceso es algo más largo. En este caso también hace uso del método `setBitMap` de la clase `ImageView` para que se muestre la imagen. Pero además, usamos un buffer de salida para guardar la imagen tomada con la cámara en la galería. Para no tener problemas con el nombre que toma la imagen la llamamos “Anti_<currentTimeinMillis>.jpg”. En la Figura 21 podemos ver el resultado de haber cargado una imagen.

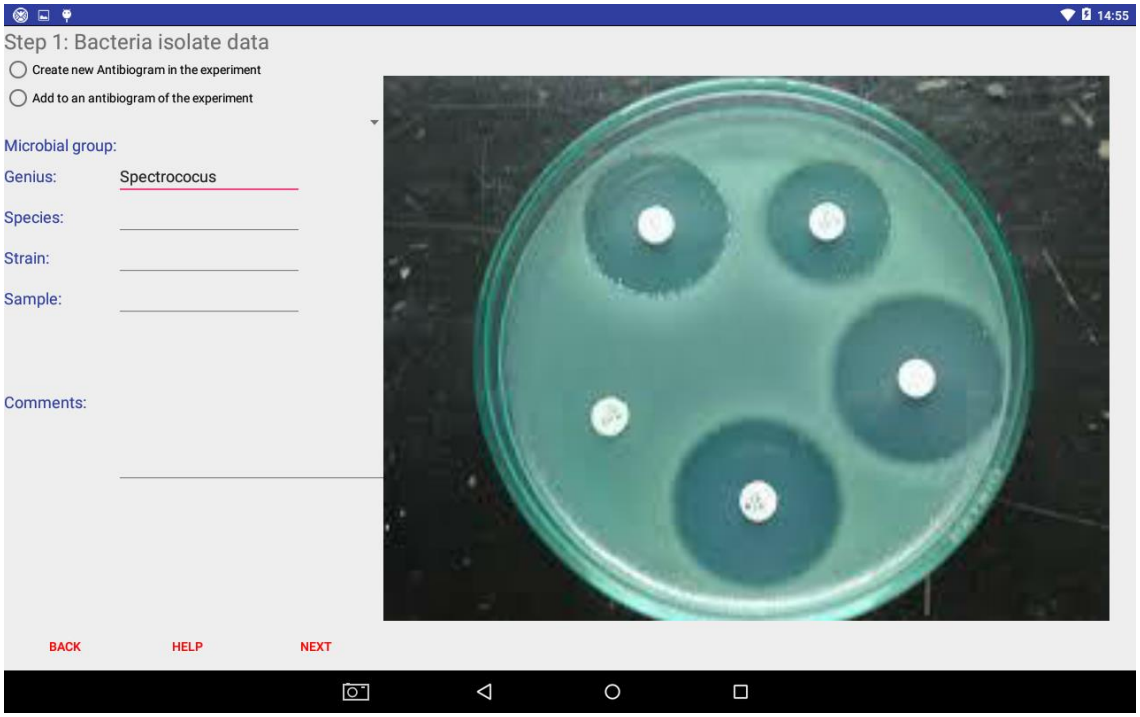

Figura 21: Imagen cargada en el ImageView

Resultados alcanzados en el sprint

En este sprint hemos realizado todas las tareas que se habían propuesto al comienzo del mismo.

Sprint 7

Tareas a realizar en este sprint.

- Estudiar las distintas formas de utilizar OpenCV en Android.
- Conectar OpenCV con nuestra aplicación Android.
- Detectar automáticamente el borde de una placa de Petri.

Desarrollo del sprint.

Llegados a este punto, necesitamos dotar a nuestra aplicación de capacidad para el tratamiento de imágenes. Como hemos comentado anteriormente, utilizaremos para ello OpenCV. Este conjunto de librerías nos permitirán, entre otras cosas, realizar la detección automática de círculos.

Lo que necesitamos hacer es conectar OpenCV con nuestra aplicación Android y para ello debemos dedicar algo de tiempo a estudiar OpenCV. Esta semana de trabajo la hemos dedicado a realizar estas dos tareas.

En primer lugar, debemos conectar Android Studio con OpenCV. Para ello, debemos optar por una de las dos siguientes posibilidades:

- Integrar OpenCV en la aplicación Android.
- Que OpenCV esté instalado en algún servidor y la aplicación Android se conecte cada vez que se necesite.

Lo primero que haremos será analizar los pros y los contras de ambas opciones para tomar una decisión. A continuación, sintetizamos las conclusiones a las que hemos llegado tras el estudio de ambas alternativas.

Ventajas de integrar OpenCV en la aplicación *vs.* utilizar un servidor

- **La aplicación no necesitaría acceso a internet.** Dado que nuestra aplicación no necesita conectarse a la red para realizar ninguna otra acción, al integrar las librerías tendríamos una aplicación que no necesita conexión de red.

- **La aplicación no dependería de un servidor externo.** De esta forma se evitan todos los posibles fallos imputables al servidor y la aplicación sería más robusta.

- **Aumentaría la velocidad.** Al no tener que esperar a la respuesta de un servidor, la aplicación respondería de forma más rápida.

Inconvenientes de integrar OpenCV en la aplicación *vs* utilizar un servidor

- **La aplicación sería algo más pesada.** Aunque no suponga mucho peso añadir las librerías de OpenCV, la aplicación ocupará más espacio. En un principio esto no parece un problema, pero al tratarse de un dispositivo móvil es algo que se debe tener en cuenta.

- **Mejoraría el mantenimiento.** Si algún día las librerías de OpenCV han cambiado y quisiéramos actualizarlas, bastaría con modificarlas en el servidor. Si las integramos en la aplicación sería necesario actualizar todas las aplicaciones.

En un principio, la dificultad de llevar a cabo cualquiera de las anteriores soluciones parece bastante similar. Quizás sea más complicada la segunda dado que se debe preocupar el programador del manejo de mensajes. La decisión que hemos tomado ha sido la primera: integrar OpenCV en la aplicación.

En esta decisión ha pesado mucho el que la aplicación necesita o no conexión de red. Dado que para el funcionamiento del resto de la aplicación no se necesita internet no hemos querido exigirlo ahora. Los otros factores, tanto positivos como negativos han tenido menos peso para tomar esta decisión. Incluir OpenCV en nuestra aplicación no la va a hacer mucho más pesada; así que, en ese sentido, no va a suponer un problema. Por otra parte, es probable que no necesitemos incluir todas las librerías de OpenCV, ya que lo que básicamente necesitamos hacer es detectar círculos automáticamente.

Mientras tratábamos de integrar OpenCV en nuestra aplicación, han surgido problemas muy diversos, algunos de los cuales creemos conveniente detallar en esta memoria.

Como ya hemos comentado OpenCV está desarrollado en C++ pero posee interfaces que permiten trabajar con estas librerías en otros lenguajes, entre ellos Java. En un primer momento, no nos dimos cuenta de este detalle y tras leer diversas fuentes en la red pensamos que era necesario instalar Android NDK. Esto lo pensamos porque cuando estábamos tratando de integrar OpenCV en Android, nos aparecía un mensaje de error que nos pedía que instalásemos Android NDK.

De acuerdo con la Android Developer Guide, Android NDK es un conjunto de herramientas que permiten implementar partes de una aplicación móvil usando código nativo como C y C++.

En la propia página para desarrolladores de Android, se avisaba de que Android NDK no era recomendado para usuarios inexpertos. Dado que no habíamos trabajado con esta tecnología anteriormente, decidimos parar la instalación y buscar algo más de información. Fue entonces cuando nos dimos cuenta de que no necesitábamos modificar las librerías de OpenCV y que podíramos hacer uso de una librería en Java.

Decidimos volver a empezar con la integración de las librerías. En todo este proceso ha sido de gran ayuda la experiencia de otros usuarios en esta labor.

Otro de los problemas que hemos tenido para integrar las librerías ha sido el tema de las versiones. Hay muchas versiones de Android SDK (24) y cada una de ellas cuenta con multitud de subversiones. Esto ya nos había dado algún que otro quebradero de cabeza, pero esta vez ha requerido mucho más tiempo.

El problema ha sido que en uno de los ficheros de configuración de OpenCV, se indicaba que se iba a utilizar una versión de Android SDK, mientras que en el fichero de configuración del proyecto se indicaba que se iba a usar otra. Android Studio se quejaba de que se estaba utilizando una versión que no estaba instalada, pero cuando accedíamos al fichero de configuración del proyecto, la versión que figuraba sí que estaba instalada. Finalmente, nos dimos cuenta que el conflicto lo generaba la versión con la que pedía trabajar OpenCV. Aunque este problema era muy sencillo nos ha llevado bastante tiempo localizar el error, cuya corrección ha resultado bastante simple.

Resultados alcanzados en el sprint

En este sprint hemos completado las dos primeras tareas planificadas. Así, nos ha quedado pendiente para su realización la detección automática del borde de una placa de Petri.

Sprint 8

Tareas a realizar en este sprint.

- Estudiar algo más sobre OpenCV.
- Implementar la detección del borde de una placa de Petri.

Desarrollo del sprint.

En este sprint tenemos como objetivo detectar la circunferencia que constituye el borde de una placa de Petri. Para ello haremos uso de las librerías de OpenCV que habíamos instalado en el sprint anterior. Antes de implementar código fue necesario estudiar algunas cosas más sobre OpenCV.

En primer lugar, vamos a hablar de la clase `Mat`. De acuerdo con [1], un objeto de esta clase puede ser usado para almacenar vectores y matrices tanto en escala de grises, como en color para representar una imagen. OpenCV hace uso de estos objetos para trabajar con imágenes; por ello, lo primero que haremos cuando queramos trabajar con imágenes en OpenCV será convertir la imagen (por ejemplo un objeto de tipo BitMap) a un objeto de este tipo. Tanto para esta opción, como para realizar la operación contraria (obtener el BitMap desde el objeto Mat) OpenCV nos ofrece métodos para hacerlo.

Una vez que tenemos el objeto Mat lo que hacemos es convertirlo a escala de grises y aplicar la función `GaussianBlur` para reducir la detección de círculos no existentes. Ahora, aplicando el método `HoughCircles` con los parámetros apropiados OpenCV detecta los círculos. Los parámetros de este método son: la imagen a analizar, el lugar donde almacenar los datos de los círculos encontrados y la distancia mínima entre los centros. Este último es muy importante, ya que un valor muy pequeño producirá detecciones no deseadas y un valor muy grande hará que no se detecten algunos círculos. Finalmente se dibujan los círculos encontrados en la imagen con ayuda de la función `circle`.

Hasta aquí, la teoría sobre cómo detectar círculos. El código que hemos implementado para ello es el siguiente.

```
public static void detectarCirculo(ImageView im){
   Bitmap bitmap=((BitmapDrawable)im.getDrawable()).getBitmap();

   Mat mat = new Mat(bitmap.getWidth(),
                     bitmap.getHeight(),CvType.CV_8UC1);
   Mat grayMat = new Mat(bitmap.getWidth(), bitmap.getHeight(),
                         CvType.CV_8UC1);

   Utils.bitmapToMat(bitmap, mat);

   int colorChannels = (mat.channels() == 3) ? Imgproc.COLOR_BGR2GRAY
              : ((mat.channels() == 4) ? Imgproc.COLOR_BGRA2GRAY : 1);

   Imgproc.cvtColor(mat, grayMat, colorChannels);

   Imgproc.GaussianBlur(grayMat, grayMat, new Size(9, 9), 2, 2);

   double dp = 1.2d;
   double minDist = 20;
   int minRadius = 0, maxRadius = 0;
   double param1 = 70, param2 = 72;

   Mat circles = new Mat(bitmap.getWidth(), bitmap.getHeight(),
                         CvType.CV_8UC1);

   Imgproc.HoughCircles(grayMat, circles, Imgproc.CV_HOUGH_GRADIENT,
```

```
                dp, minDist, param1, param2, minRadius, maxRadius);

  int numberOfCircles = (circles.rows() == 0) ? 0 : circles.cols();

  for (int i=0; i<numberOfCircles; i++) {
    double[] circleCoordinates = circles.get(0, 0);
    int x = (int) circleCoordinates[0];
    int y = (int) circleCoordinates[1];

    Point center = new Point(x, y);
    int radius = (int) circleCoordinates[2];
    Imgproc.circle(mat, center, radius, new Scalar(0, 255, 0), 4);

    Imgproc.rectangle(mat, new Point(x - 5, y - 5),
                new Point(x + 5, y + 5), new Scalar(0, 128, 255), -1);
  }

  Utils.matToBitmap(mat, bitmap);
  im.setImageBitmap(bitmap);
}
```

Parece sencillo, pero cuando hemos ejecutado nuestra aplicación ha empezado a dar diversos problemas. Hemos conseguido resolver algunos que tenían que ver con la obtención del BitMap desde el ImageView. Sin embargo, nos ha dado también un problema de librería en tiempo de ejecución. Cuando precargamos las librerías en la aplicación salta una excepción y se detiene la aplicación. A continuación, mostramos una captura de la excepción que salta.

```
Caused by: java.lang.UnsatisfiedLinkError: Couldn't load opencv_java310: findLibrary returned null
    at java.lang.Runtime.loadLibrary(Runtime.java:365)
    at java.lang.System.loadLibrary(System.java:535)
    at ImageWork.CircleDetect.<clinit>(CircleDetect.java:27)
```

El problema es que el programa se queja de que las librerías no están. Sin embargo, sí que están. Hemos estado consultando muchas fuentes por internet y tratado de seguir los consejos que allí nos daban para poder solucionar el problema. Como no fuimos capaces de solucionarlo decidimos integrar las librerías de otra forma.

Lo que vamos a hacer es utilizar ficheros .jar que están accesibles a través de GitHub [9] en el proyecto Javacv de Bytedeco. Siguiendo los pasos que allí se indican, las incluimos en el proyecto. Por desgracia, las funciones que estas librerías nos proporcionan no son tan completas como las que teníamos anteriormente. En consecuencia, tendremos que modificar el programa. Con esta labor comenzaremos el próximo sprint.

Resultados alcanzados en el sprint

Debido a los problemas que hemos comentado, no hemos sido capaces de implementar la detección del borde una placa de Petri.

Sprint 9

Tareas a realizar en este sprint.

- Estudiar JavaCV
- Implementar la detección del borde de una placa de Petri con JavaCV.
- Implementar la opción de recortar una imagen tras la detección del borde de la placa de Petri.
- Implementar la opción de deshacer el recorte de una imagen.

Desarrollo del sprint.

Cuando trabajamos con la librería JavaCV, en lugar de utilizar un objeto Mat para representar una imagen, utilizaremos objetos de la clase `IplImage`. Lo primero que hacemos es obtener el objeto IplImage desde el BitMap de nuestro ImageView. A continuación, convertimos esta imagen a escala de grises. Usando el método estático `cvSmooth` para suavizar los bordes, ayudamos en la mayor parte de los casos a una buena detección de los círculos.

Con el método `cvHoughCircle` detectamos los círculos que aparecen en la imagen. Debemos elegir adecuadamente los parámetros que le pasamos a este método. Al radio mínimo le damos un valor grande para evitar detectar círculos pequeños, al radio máximo le damos un valor muy alto para poder considerar círculos muy grandes y a la distancia entre centros le damos un valor muy grande para que no detecte más de un círculo. Los círculos detectados los almacenamos en un objeto de tipo `CVSeq`.

Haciendo uso del método `cvGetSeqElem` extraemos cada uno de los círculos almacenados en el objeto `CVSeq` (en este caso sólo habrá uno). Dibujamos cada circunferencia, obteniendo para ello su centro, un objeto de tipo `CvPoint` y su radio. Haciendo uso del método `cvCircle` dibujamos la circunferencia.

Una vez dibujada la circunferencia, debemos convertir el objeto IplImage en un BitMap de nuevo. Una vez que lo hemos convertido, asignamos al ImageView el nuevo BitMap. En la Figura 21 se muestra una captura del resultado de este proceso.

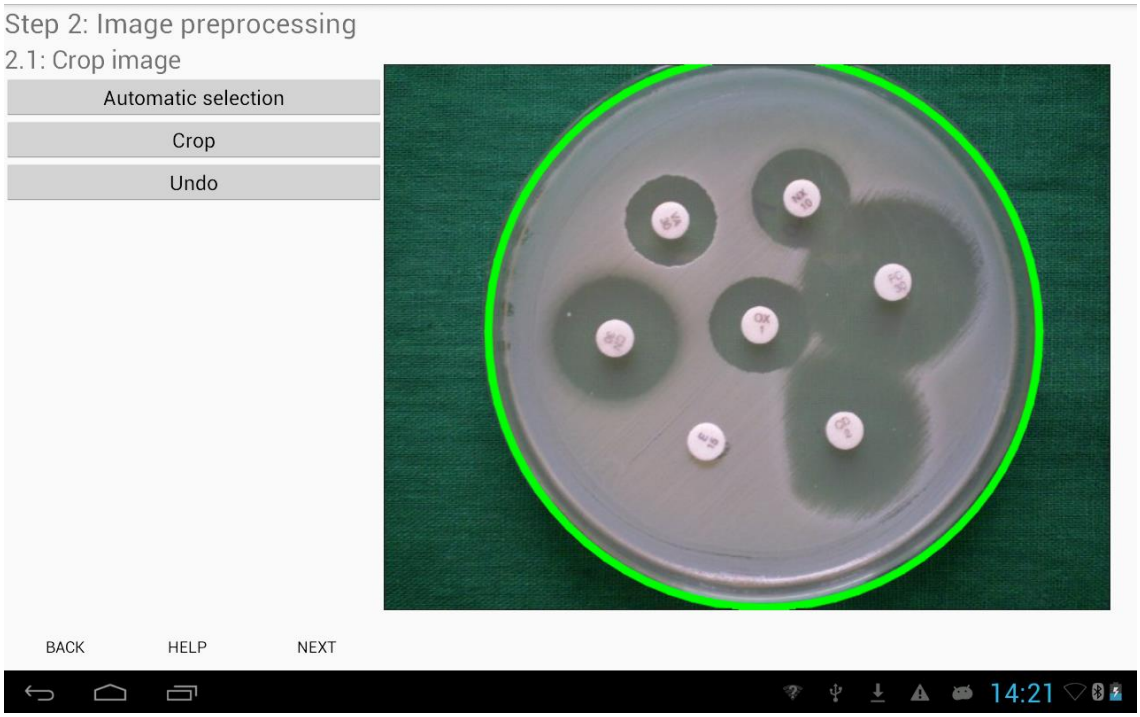

Figura 21: Detección del borde de una placa de Petri

Pese a que la aplicación es capaz de detectar el borde de la placa de Petri en la mayor parte de las imágenes, todavía falla en algunas. Para no seguir estancados en este punto nos ponemos ahora con la opción de cortado de la imagen. Más adelante deberemos abordar el problema de la detección del borde en algunas imágenes.

Con respecto al cortado de la imagen el proceso es *a priori* muy sencillo. Se pretende obtener un rectángulo que se ajuste al círculo detectado, despreciando de esta manera la parte de fuera de la placa de Petri. Conocido el centro y el radio del círculo, podemos calcular cuáles son las dimensiones y la posición del rectángulo que queremos quedarnos.

Lo que hacemos es seleccionar la región de interés, con el método `cvSetImageROI`, en la imagen. A continuación, se copian los bytes de esta sección a un nuevo objeto IlpImage. Obtenemos el bitMap correspondiente y lo colocamos en el ImageView. Aunque parece muy sencillo nos está causando algún problema y, por algún motivo todavía desconocido, nos está devolviendo un bitMap vacío.

Tras unas horas buscando el error hemos caído en la cuenta de que no necesitábamos la librería JavaCV para cortar la imagen, ya que con los propios métodos de BitMap nos bastaba. Hemos corregido el programa y ha resultado muy sencillo, basta con usar el método estático `createBitMap` de la clase `BitMap`. En la Figura 22 mostramos el resultado de cortar la imagen.

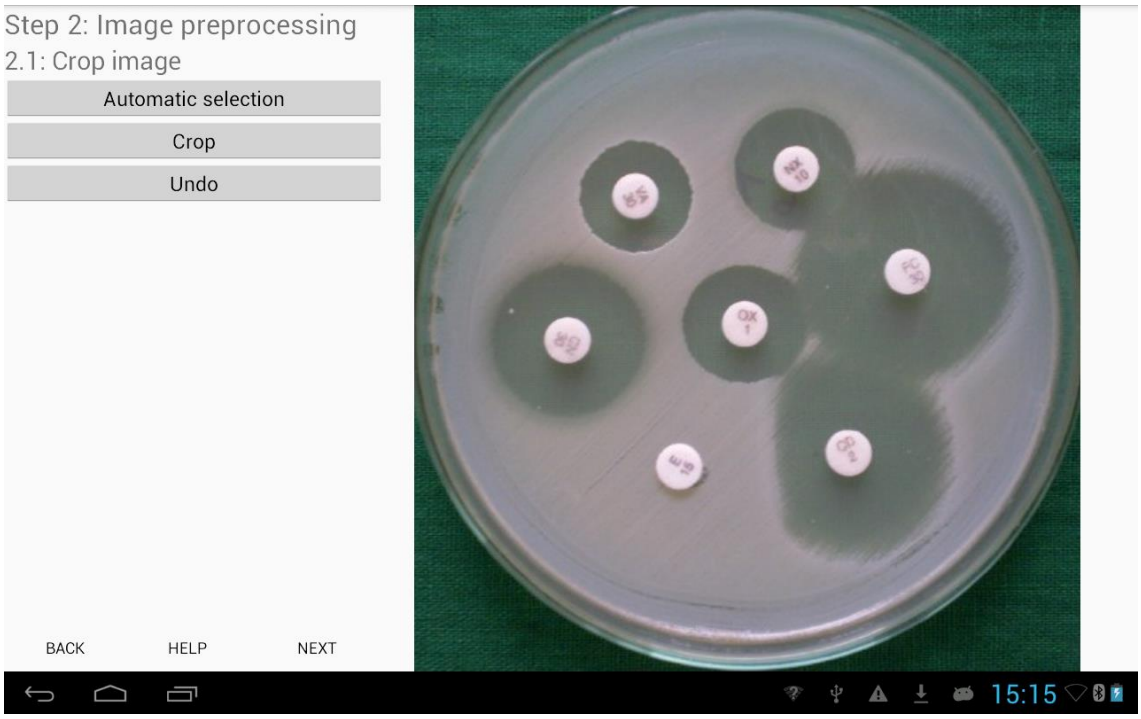

Figura 22: Recorte de la imagen

Resultados alcanzados en el sprint

En este sprint hemos realizado todas las tareas que se habían propuesto salvo la funcionalidad del botón de deshacer. Sin embargo, queda pendiente mejorar la detección del borde de una placa de Petri, ya que hay algunas imágenes en las que todavía no funciona.

Sprint 10

Tareas a realizar en este sprint.

- Implementar la opción de deshacer el recorte de una imagen.
- Mejorar la detección automática del borde de una placa de Petri.

Desarrollo del sprint.

En primer lugar, veamos brevemente la funcionalidad del botón de deshacer. Al hacer clic en este botón se debe mostrar la imagen original previa al corte. Para implementarlo basta con guardar la imagen original y cambiar el bitMap que se muestra sobre el ImageView en cada momento.

Una vez que los tres botones están implementados, hay que controlar el momento en el que cada uno puede ser pulsado. En un primer momento solo se puede pulsar el botón de detección automática. Una vez que este ha sido pulsado, solo se puede pulsar el botón de cortar. Finalmente, cuando este ha sido pulsado, solamente se puede pulsar el botón de deshacer.

En el sprint anterior hemos estado trabajando con el método `HoughCircle` para detectar círculos. Como hemos comentado, recibe muchos parámetros que condicionan la detección. Hasta ahora no les habíamos prestado demasiada atención, pero va a ser necesario estudiarlos en profundidad.

- Objeto de tipo `IlpImage` que contiene la imagen, en escala de grises, con la que se va a trabajar.
- Objeto de tipo `CvMemStorage` para almacenar el resultado en memoria.
- Entero para identificar al método con el que trabajar, nosotros trabajamos con el método `CV_HOUGH_GRADIENT`.
- Entero que representa la ratio inversa. En nuestro caso 1.
- Entero que representa la distancia mínima entre dos centros de dos círculos.
- Entero que representa el umbral de error para el detector de borde del círculo.
- Entero que representa el umbral de error para el detector del centro del círculo.
- Entero que representa el radio máximo de un círculo.
- Entero que representa el radio mínimo de un círculo.

El primero de los parámetros es un objeto IlpImage. El método `HoughCircle` trabaja con objetos de este tipo, que además deben ser imágenes en escala de grises, en 8bits y de canal simple. Por este motivo, antes de invocar el método `HoughCircle`, lo que hacemos es transformar nuestra imagen a escala de grises.

Los parámetros referidos a los umbrales no son de momento muy importantes. Cuando tratemos de detectar los halos formados alrededor de las pastillas de antibiótico jugarán un papel crucial. Ahora nos centraremos en la distancia entre centros y los valores mínimo y máximo que el radio de un círculo pueda tomar. Estos tres parámetros miden números de píxeles.

Lo primero que observamos es que las imágenes con las que la aplicación va a trabajar pueden tener un tamaño en píxeles muy diferente. Podemos encontrar imágenes de poca precisión, es decir, con pocos píxeles e imágenes de mucha precisión y, por tanto, de muchos píxeles.

En un primer momento nuestra aplicación solo era capaz de detectar círculos en unas pocas imágenes. Observamos que el problema era que estábamos ajustando demasiado los valores mínimo y máximo para la detección de círculos. Al aumentar el radio máximo y reducir el radio mínimo, nuestra aplicación fue capaz de detectar el borde de la placa en todas las imágenes de las que disponíamos. Sin embargo, esto tuvo un efecto negativo, la aplicación tardaba ahora mucho más tiempo en detectar los círculos. Así, nos dimos cuenta de que hay una dependencia entre la eficiencia con la que la aplicación detecta el borde la placa y la precisión de las imágenes (entendiendo la precisión como el número de píxeles que tiene una imagen) que la aplicación es capaz de detectar.

Una buena forma de guardar un equilibrio entre la eficiencia y el número de imágenes con las que la aplicación es capaz de trabajar, es elegir los valores mínimo y máximo de los radios en función del número de píxeles de una imagen. De esta forma, el radio máximo será el 50% del máximo entre el ancho y el alto de la imagen. Por supuesto, estos valores medidos en número de píxeles. Para el radio mínimo hemos tomado el

20% del mínimo entre el ancho y el alto de la imagen. Y para la distancia entre centros hemos tomado el mismo valor que para el radio máximo ya que solo queremos detectar un círculo.

Otro tema que afecta a la eficiencia de la aplicación es la elección de umbrales de detección. Si damos valores altos, la aplicación tarda menos en detectar el borde pero falla en más imágenes. Si damos valores bajos, la aplicación detecta el borde en cualquier imagen, pero tarda más. De nuevo hemos ajustado los valores de forma que la eficiencia no se vea muy afectada y que la aplicación funcione con todas las imágenes.

Resultados alcanzados en el sprint

Hemos completado las tareas planificadas para este sprint. Además, al ejecutar ahora nuestra aplicación, funciona con todas las imágenes de las que disponemos y emplea menos tiempo en detectar el borde de la placa del que empleaba antes.

Sprint 11

Tareas a realizar en este sprint.

- Añadir una barra de progreso durante la detección del borde de una placa de Petri.
- Implementar la detección de los discos de antibióticos de un antibiograma.

Desarrollo del sprint.

Pese a que hemos sido capaces de reducir el tiempo de detección del borde de la placa de Petri, este sigue siendo alto. Es tan alto, que consideramos necesario incluir algún tipo de barra de progreso mientras trabaja la aplicación. De esta forma, el usuario se dará cuenta de que la aplicación está trabajando en la detección, y que, por lo tanto, debe esperar.

Siguiendo las indicaciones que encontramos en la Android Developer Guide tratamos de añadir la barra de progreso. En este caso, optaremos por una barra de progreso de tipo determinado, en la que se muestre el porcentaje de trabajo que se ha realizado hasta el momento.

Para programar la barra de progreso debemos hacer uso de hilos de ejecución. Uno de los hilos se encarga de la barra de progreso y otro, el principal, de la detección del borde de la placa. En la Figura 23 mostramos una captura de este paso.

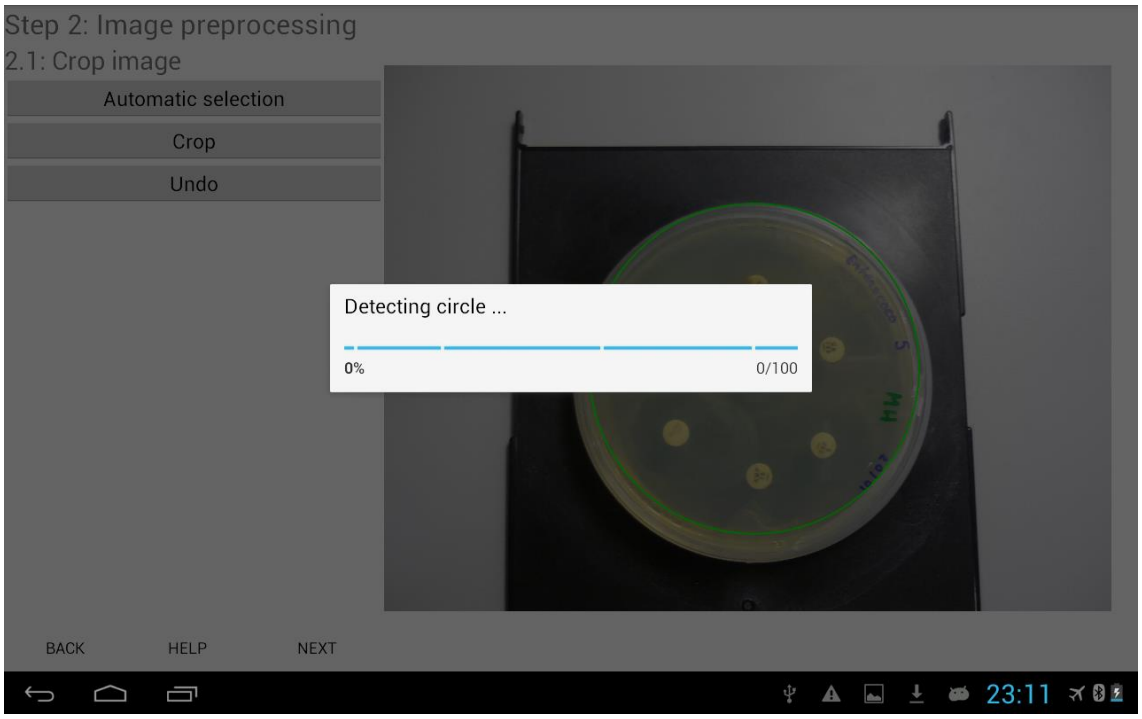

Figura 23: Barra de progreso mientras se detecta el borde

Una vez que ya somos capaces de detectar el borde de la placa de Petri y recortar la imagen, toca detectar los discos de antibiótico situados en la placa. Para ello haremos uso también del método `HoughCircle`, pero debemos modificar los parámetros. En este caso el radio mínimo será muy pequeño y el radio máximo será mucho menor que antes. Además la distancia entre centros permitirá que en la imagen se puedan detectar varios círculos.

Sin embargo, los parámetros que ahora van a jugar un papel fundamental son los umbrales de error para el detector del borde y del centro del círculo.

Si ponemos un umbral muy bajo, se producirán falsos positivos y, si ponemos un umbral muy alto, muchos de los discos quedarán sin detectar. Por lo tanto, debemos elegir estos valores cuidadosamente para que se mantenga el equilibrio. En cualquier caso, habrá imágenes en las que la aplicación no será capaz de localizar los discos de antibiótico sin dar algún falso positivo. En estas ocasiones, habrá que decidir qué es lo preferible, que el usuario tenga que marcar los círculos que faltan o que marque los que sobran. De igual forma que se ha hecho en la versión de escritorio, optaremos por lo primero. Es preferible que se quede algún disco sin detectar a que se produzcan falsos positivos.

En la Figura 24, mostramos una captura en la que se aprecia cómo se producen falsos positivos si ponemos unos valores demasiado bajos a los umbrales. En la Figura 25, añadida a continuación de la anterior, mostramos cómo no se detectan todos los círculos para valores demasiado altos de los umbrales.

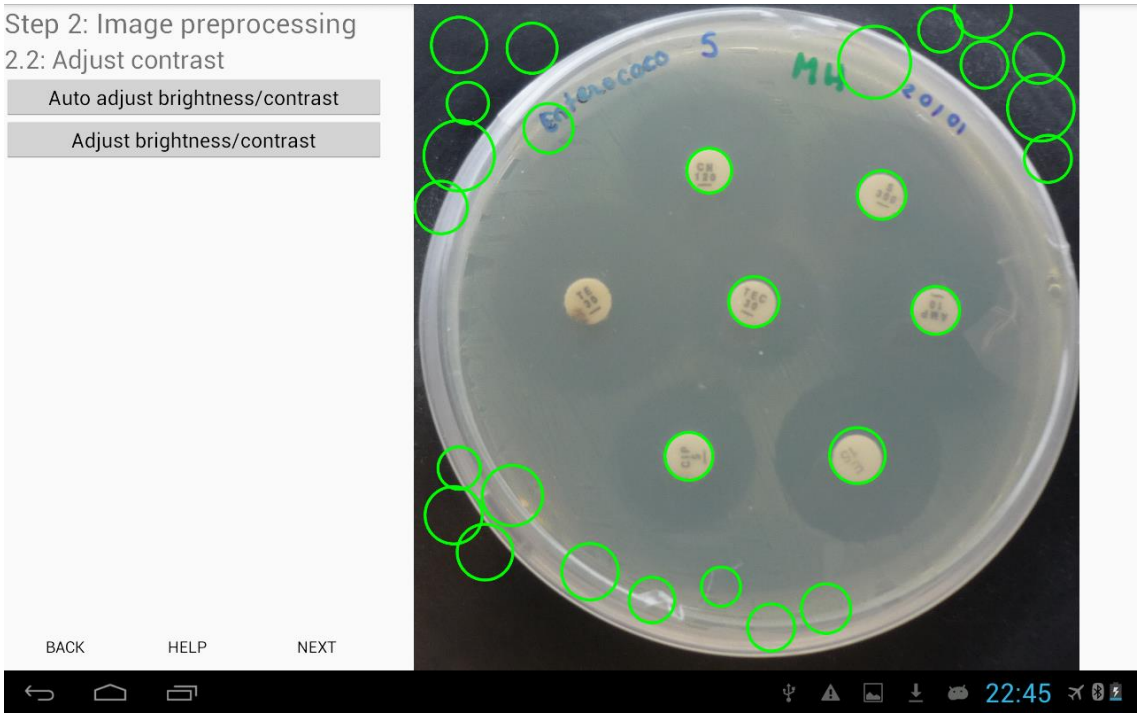

Figura 24: Detección círculos para valores bajos de los umbrales.

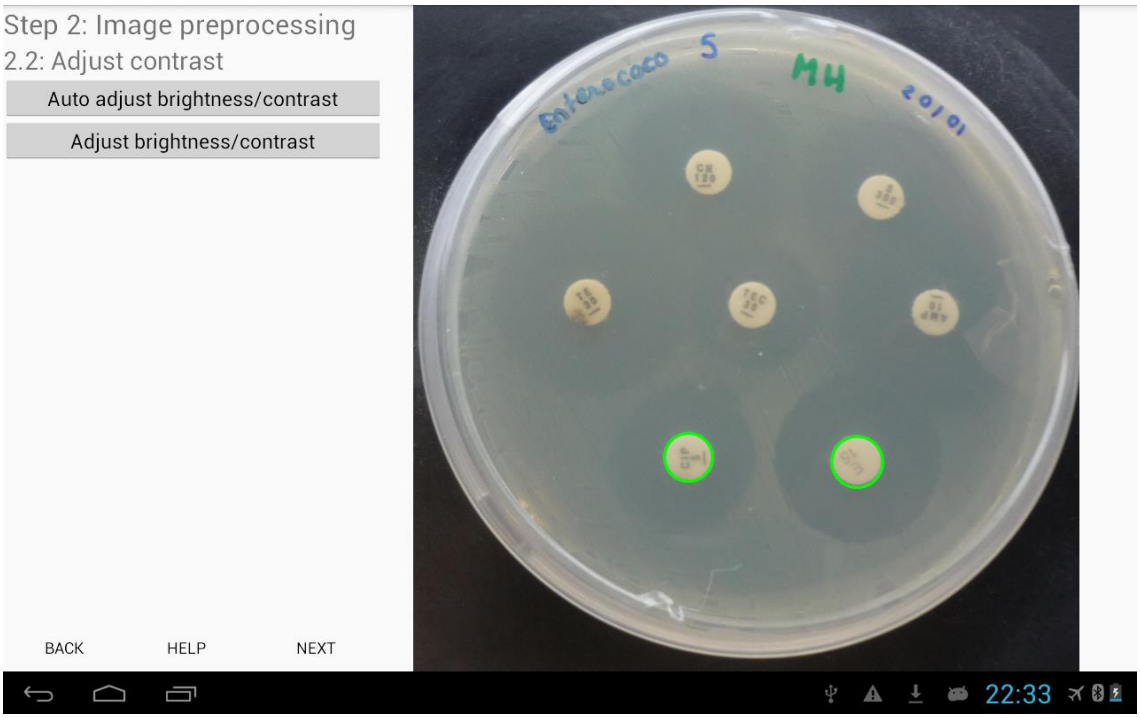

Figura 25: Detección círculos para valores altos de los umbrales.

Resultados alcanzados en el sprint

En este sprint hemos completado las tareas planificadas al comienzo. Sin embargo, la barra de progreso nos está dando algún que otro problema en las últimas pruebas. Por motivos que aún no conocemos, en algunas pruebas se ha quedado bloqueada una vez que la aplicación ya había terminado con la detección. Como no es de prioridad alta, la vamos a desactivar y seguiremos con otras tareas.

Sprint 12

Tareas a realizar en este sprint.

- Dibujo y medida del diámetro de un disco de antibiótico.
- Ajuste automático del brillo de una imagen.
- Inserción manual de circunferencias para los discos de antibióticos que no se hayan detectado automáticamente.

Desarrollo del sprint.

Una vez que ya se han detectado los discos de antibiótico, el usuario tiene la opción de añadir manualmente los que falten por detectar o de eliminar o modificar alguno de los discos ya detectados. Esta función la dejamos para más adelante, ya que para implementarla no utilizaremos OpenCV. Así pasamos ahora a medir el diámetro de uno de los discos de antibiótico, para lo que sí que hacemos uso de OpenCV. Esto es necesario para permitir que la aplicación pueda relacionar los valores que mide en la imagen (en píxeles) con los valores que están contenidos en las tablas EUCAST (en milímetros).

Es importante notar que siempre habrá al menos un disco de antibiótico detectado, o introducido manualmente, ya que en otro caso la aplicación no habrá permitido superar el paso 3.1. Por tanto, siempre podremos dibujar el diámetro de un disco.

Para llevar a cabo esta operación, se muestra al usuario la placa de Petri con el diámetro de uno de los círculos dibujado. El usuario debe introducir la medida real de uno de los discos de antibiótico. Por defecto, el valor que introduce la aplicación es de 6mm. En la Figura 26 mostramos una captura de pantalla de este paso.

Figura 26: Medida del diámetro de uno de los discos de antibiótico

Como vemos en la anterior imagen, además del campo donde introducir la medida real del disco de antibiótico, hay dos botones que permiten borrar o añadir una línea. También hay una barra para aumentar o reducir el tamaño de dicha línea. La funcionalidad de estos controles no está implementada de momento.

Una vez que el diámetro de uno de los discos ha sido introducido, el siguiente paso es leer el código que identifica cada disco de antibiótico y que está escrito en cada uno de los discos. Para ello, lo primero que debemos hacer es generar dinámicamente el botón y el campo de texto en el que aparecerá el nombre de cada disco. Para lograrlo, haremos uso del constructor que implementamos anteriormente, cuando estábamos desarrollando la interfaz de la aplicación y que nos permite generar un Fragment con el número de controles que deseemos. En este caso, el número es la longitud de la secuencia de círculos que identifican a cada disco de antibiótico.

En la Figura 27 mostramos una captura de pantalla de este paso. En la imagen podemos ver como en este caso se han detectado 5 discos de antibiótico y por tanto se generan 5 controles. Los discos de antibiótico aparecen numerados.

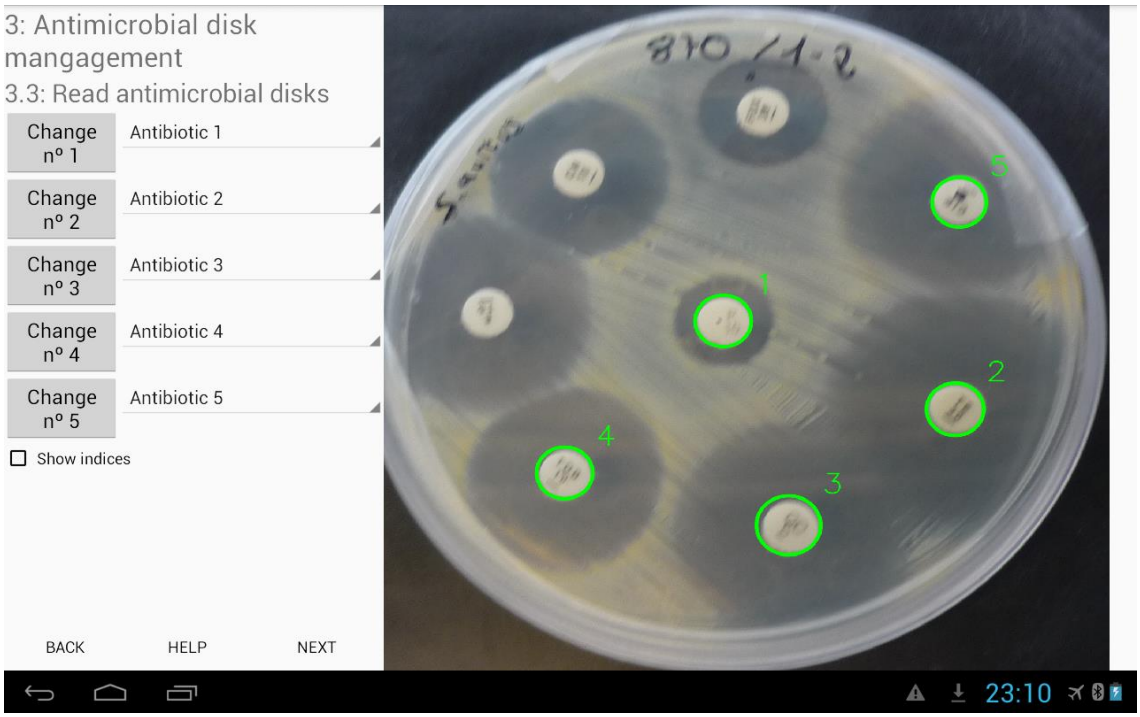

Figura 27: Generación de los controles para cada disco de antibiótico

Ahora vamos a volver un poco atrás para implementar una función que antes pasamos por alto. El paso 2.1 de la aplicación permite un ajuste del brillo y del contraste de la imagen. En su momento, lo dejamos como tarea pendiente dado que su prioridad era más baja. Hemos decidido hacerlo en las últimas horas de este sprint para no empezar con una tarea más compleja, ya que requeriría más horas de las que nos restan. Tras investigar un poco por la red, hemos encontrado a un usuario que nos enseña a modificar el brillo y el contraste de un Bitmap de una forma muy sencilla. Creamos un método que recibe tres parámetros: el Bitmap con el que trabajar, el nuevo valor del

brillo que tendrá la imagen y el nuevo valor del contraste que tendrá la imagen. El entero que representa el brillo estará comprendido entre -255 y 255 y el entero que representa el contraste estará comprendido entre 1 y 10.

Una vez que tenemos el método implementado es muy sencillo implementar la funcionalidad del botón de ajuste automático del brillo y del contraste. Hemos decidido que los valores que le vamos a pasar por defecto a esta función son 1 para el contraste y 40 para el brillo. Por defecto estos valores en una imagen valen 1 y 0 respectivamente. Por lo tanto, lo que hacemos es aumentar un poco el brillo de la imagen. En la Figura 28 podemos ver el efecto que aplicar el ajuste automático tiene sobre una imagen. A la izquierda, mostramos la imagen sin el ajuste automático de brillo y contraste y, a la derecha, el resultado de aplicarlo.

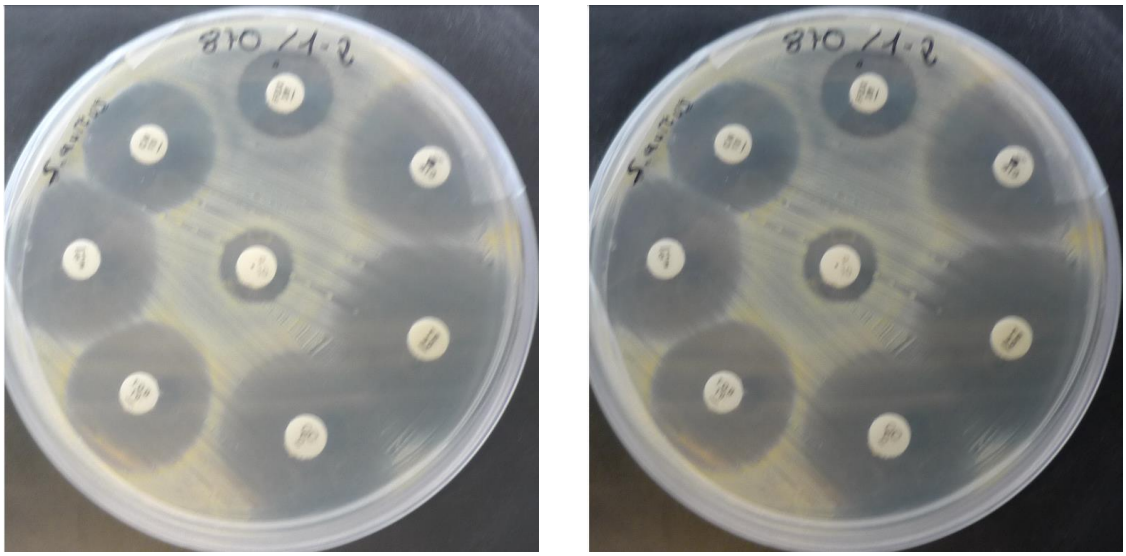

Figura 28: Resultado del ajuste automático del brillo y el contraste

Implementar el ajuste manual del brillo y del contraste no resultará muy complicado ahora que ya disponemos de un método que aplica nuevos valores para el brillo y para el contraste. En cualquier caso, al igual que venimos haciendo con toda la funcionalidad que supone un ajuste manual, la dejamos para más adelante.

Resultados alcanzados en el sprint

En este sprint nos ha faltado realizar la inserción manual de circunferencias para los discos de antibióticos no detectados.

Sprint 13

Tareas a realizar en este sprint.

- Reunión con el cliente para priorizar tareas.
- Inserción manual de circunferencias para los discos de antibióticos que no se hayan detectado automáticamente.

Desarrollo del sprint.

Dado que nos quedan las últimas 60 horas de trabajo, hemos decidido mantener una reunión con el cliente con el objetivo de priorizar las tareas. Al término de esta reunión hemos acordado que las dos tareas más importantes que quedan por hacer son: permitir la inserción manual de círculos para añadir discos de antibióticos y detectar automáticamente los halos que se formen alrededor de cada disco.

Empezamos con la inserción manual de los círculos. Para ello, tendremos que utilizar algo nuevo hasta el momento: usaremos la clase `Canvas` de Android. Antes de empezar a implementar la funcionalidad, hemos hecho una búsqueda para ver si encontrábamos por la red algo que nos pudiera servir. Hemos encontrado un fragmento de código que permitía dibujar círculos opacos de radio aleatorio sobre un fondo blanco que nos ha sido de gran ayuda.

Tal y como nosotros queríamos, en este código se hace uso de la clase `Canvas` de Android. De acuerdo con la Android Developer Guide, la clase `Canvas` contiene las llamadas para dibujar. Para poder dibujar una figura necesitamos 4 elementos: el BitMap sobre el que dibujar, un lienzo para albergar la escritura en el mapa de bits, el dibujo primitivo y un objeto de tipo Paint que indicará con qué vamos a dibujar. A este último objeto le podemos indicar el color, el grosor, etc. Para hacer uso de Canvas hemos implementado una clase que hemos llamado `CircleDrawingView`, que extiende de `ImageView` y que implementa los métodos `onTouch` y `onDraw`.

En nuestro caso, le pasaremos como BitMap el de la ImageView sobre la que trabajamos, como lienzo un objeto de tipo Rect de iguales dimensiones que el BitMap. El dibujo primitivo será para nosotros un círculo, y la pintura indicará el color y el grosor, si el círculo tiene relleno o no (en nuestro caso no, ya que solo nos interesa el borde).

Ahora el usuario puede dibujar un círculo y arrastrarlo hasta situarlo sobre alguno de los discos de antibiótico ya detectados. Hemos considerado que lo más cómodo es evitar que el usuario tenga que reajustar el radio del círculo. Para ello, construimos círculos de tamaño fijo, cuyo radio es la media de los radios de los discos de antibiótico que ya se han detectado automáticamente. Así un paso previo antes de dibujar el círculo es obtener la media de los radios de los círculos que ya se habían detectado. En la siguiente imagen podemos ver esta situación. Cuando el usuario toca sobre la imagen aparece un círculo, que podrá arrastrar para situarla donde mejor le parezca.

Figura 29: Al tocar la imagen aparece un círculo que se puede arrastrar.

Una vez que ya somos capaces de dibujar los círculos, hay que guardar los datos de cada círculo (posición y radio) para poder trabajar luego con ellos. Después que el usuario ha situado el círculo sobre un disco de antibiótico, puede hacer clic sobre el botón “Add disk” para que se añada el disco. En este punto hemos tenido un problema. Como hasta ahora los círculos que se detectaban automáticamente, los dibujábamos con las funciones de la librería OpenCV. Hemos intentado dibujar los círculos que estaban dibujados con Canvas con las funciones de OpenCV, pero el resultado no era el esperado.

Según la documentación de OpenCV y la documentación de Canvas, en ambos casos se dan las medidas en píxeles, por lo que pasar los datos de un sistema a otro debía ser trivial. Sin embargo, lo que hemos observado es que esto no ocurría. Cuando intentábamos dibujar con las funciones de OpenCV el círculo que el usuario había situado y cuyos datos teníamos en Canvas, observábamos que el círculo no se situaba donde debía. En la siguiente imagen podemos ver este fenómeno. El círculo blanco de arriba a la izquierda debía haber sido dibujado donde vemos el círculo azul.

Figura 30: En azul vemos el círculo dibujado con Canvas; en blanco, el círculo dibujado con OpenCV para los mismos valores de radio y centro.

Tras leer algo de documentación de OpenCV hemos recordado que uno de los parámetros de las funciones de dibujar indica el número de subdivisiones que se aplica a cada píxel. Lo hemos ajustado para que no se subdividiera, pero el problema persistía. Haciendo alguna que otra prueba, hemos observado que aproximadamente los valores de un mismo círculo, valían la mitad en OpenCV que en Canvas. Sin embargo, esto solo ocurría en determinadas imágenes.

En el siguiente sprint, empezaremos guardando los círculos dibujados haciendo uso de las funciones de Canvas para esquivar el problema de la conversión de valores entre Canvas y OpenCV.

Resultados alcanzados en el sprint

Al término del sprint nos queda pendiente implementar el guardado de la imagen con la nueva circunferencia dibujada. Esto será la funcionalidad del botón “Add disk”.

Sprint 14

Tareas a realizar en este sprint.

- Implementar el guardado de la circunferencia dibujada por el usuario, es decir, la funcionalidad del botón “Add disk”.
- Implementar la primera parte del paso 4, consistente en la detección automática del halo formado alrededor de los discos de antibiótico.

Desarrollo del sprint.

Como comentábamos al término del sprint anterior, debemos implementar la funcionalidad del botón “Add disk” para poder añadir un disco de antibiótico una vez que el usuario ha situado manualmente un círculo sobre este. Para ello vamos a implementar un método que llamaremos `guardarDibujo` dentro de la clase `CircleDrawingView` que estamos implementado. Este método se invocará cuando el usuario pulse sobre el botón “Add disk”.

Aunque su funcionalidad es bastante sencilla nos ha dado muchos problemas. Lo que finalmente hemos hecho para implementar este método es crear un objeto Canvas con el que dibujar pasándole el BitMap del ImageView. A continuación, hacemos uso de la función `drawCircle` de `Canvas` pasándole un objeto Paint, el radio y la posición del círculo.

Además, cuando el usuario añada círculos guardaremos sus datos ya que nos están indicando nuevos discos de antibiótico con los que posteriormente tendremos que trabajar. Ahora sí; el usuario ya puede añadir manualmente la posición de nuevos discos de antibiótico.

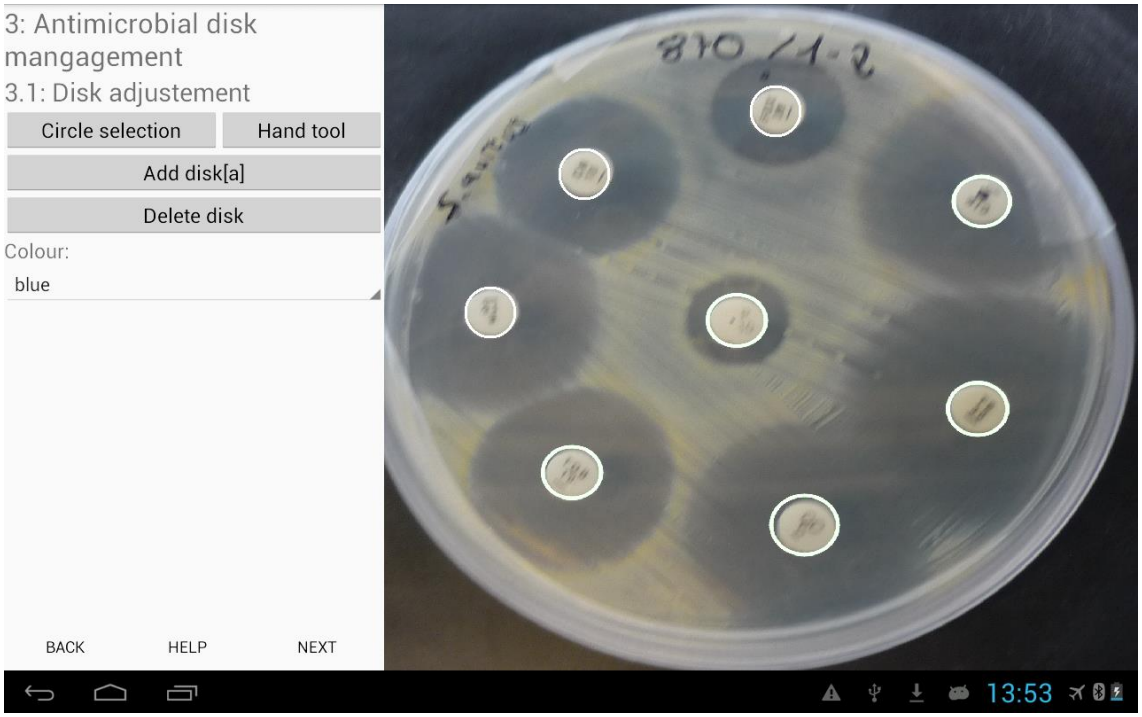

Figura 31: Misma captura que en la Figura 30 pero esta vez con todos los discos ya introducidos por el usuario.

Ahora toca ponerse con la detección de los halos formados alrededor de estos círculos. Como paso previo a esto está la lectura del código de cada disco o inserción manual del tipo de antibiótico que se usa (Paso 3.3). Este paso lo implementaremos más adelante si nos queda tiempo, ya que así lo acordamos con el usuario.

Ahora nos ponemos con la detección de los halos. Para ello, contamos con una ventaja respecto a la detección de los círculos: conocemos el centro de los discos. Por tanto, el problema se reduce a detectar el radio del halo.

Hemos estado buscando entre la documentación de OpenCV en busca de algún método que nos permitiera detectar círculos de centro conocido. Sin embargo, en contra de nuestras expectativas, no hemos encontrado ninguno. Por ello, lo que vamos a hacer es detectar círculos en toda la imagen con un umbral bajo, de forma que se detecten el mayor número de círculos posibles.

A continuación, implementaremos un método que decida si un círculo es concéntrico a un disco o no. Por tanto, no nos preocupa que en la anterior detección haya falsos positivos, ya que si se detecta un círculo cuyo centro no está sobre un disco lo desecharemos. Esta función será en cualquier caso demasiado lenta. Seguimos pensando que debiéramos ser capaces de aprovechar el hecho de conocer el centro.

En la siguiente imagen mostramos una captura de pantalla resultante de haber ejecutado la detección de los halos. Como podemos observar solamente ha sido capaz de detectar el halo de uno de los discos. En el próximo sprint trataremos de mejorar esta funcionalidad.

Figura 32: Detección del halo de los discos de antibiótico.

Resultados alcanzados en el sprint

En este sprint hemos logrado implementar la funcionalidad del botón “Add disk” satisfactoriamente. Además, hemos implementado la detección del halo formado alrededor de un disco. Sin embargo, en esta última parte queda trabajo por hacer, ya que no detecta bien muchos de los halos.

Sprint 15

Tareas a realizar en este sprint.

- Implementar otra función para detectar los discos de antibióticos.
- Implementar otra función para detectar los halos formados alrededor de un disco de antibiótico ya detectado.
- Valorar y redactar las conclusiones de este trabajo.

Desarrollo del sprint.

En este último sprint vamos a intentar dar otra implementación a la función que permite detectar los halos. Además, intentaremos también dar otra implementación para la función que detecta los discos de antibiótico. Así, podremos seguir ampliando nuestro conocimiento sobre OpenCV al mismo tiempo que tratamos de mejorar estas funcionalidades. Contamos con la ayuda del código utilizado en la aplicación de escritorio que está escrito en Python.

Para detectar los discos de antibióticos, buscamos zonas de la imagen que sean blancas, ya que estos discos son siempre de este color. Para ello, usamos la escala de colores HSV (matiz, saturación y valor). De estos tres canales, el que resulta más interesante es el tercero, que recoge la luminosidad de la imagen. Utilizamos un método de OpenCV para quedarnos con este canal.

Después, aplicamos un threshold alto para quedarnos con las partes más luminosas de la imagen, es decir, los discos de antibiótico. A continuación, detectamos los contornos y comprobamos si son circunferencias o no. Sobre la imagen original dibujamos las circunferencias encontradas.

Para la detección de los halos el proceso es diferente. Como ya conocemos dónde se sitúan los discos de antibiótico, lo que hay que hacer es buscar círculos concéntricos. Para ello, lo que hacemos es que, partiendo desde el borde del disco de antibiótico, comprobamos si un círculo un poco más grande (de 5 píxeles más) sigue teniendo el mismo color. Si tiene el mismo color, probamos con uno más grande, si el color es distinto, eso significa que ya estamos en el borde del halo. Una vez detectado el halo se dibuja sobre la imagen.

El nuevo algoritmo para la detección de los discos de antibiótico funciona muy bien en imágenes en las que los discos son más blancos ya que precisamente busca las zonas de este color. En la Figura 33 podemos ver un resultado de esto mismo.

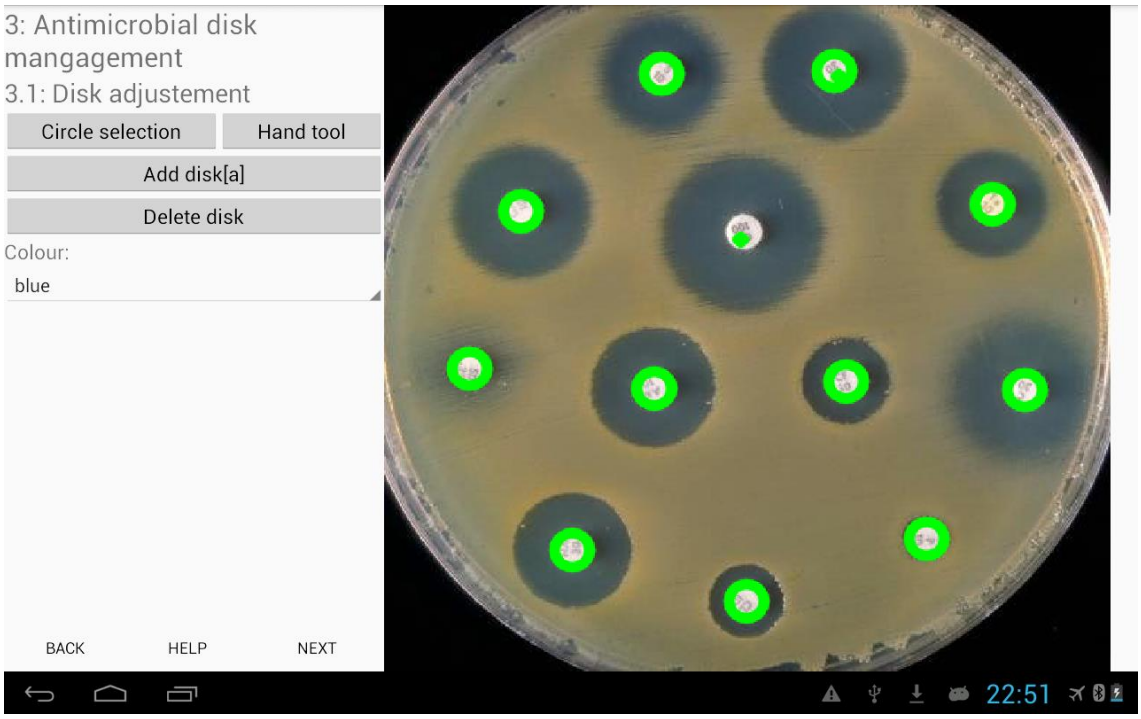

Figura 33: Detección de los discos de antibiótico buscando zonas blancas.

Además, en este sprint también hemos mejorado el algoritmo que utilizábamos antes para la detección de los discos de antibiótico a través de la búsqueda de círculos. En la siguiente Figura vemos como para la imagen que aparece funciona perfectamente. En la Figura 27 se puede ver como se dejaba de detectar 3 discos de antibiótico en esta misma imagen.

Figura 34: Detección de discos de antibiótico buscando círculos.

Resultados alcanzados en el sprint

En este sprint hemos completado las tareas propuestas. Hemos ampliado nuestros conocimientos sobre OpenCV estudiando otros nuevos algoritmos. Pese a que no hemos llegado a ejecutar sin problemas el algoritmo para la detección de los halos, estamos satisfechos con lo aprendido. Además, hemos meditado sobre las conclusiones de este trabajo.

Conclusión

Con la realización de este trabajo, he tenido la suerte de trabajar con distintas tecnologías informáticas que hasta el momento eran desconocidas para mí. Por un lado, he tenido una primera experiencia en programación móvil, ya que hasta el momento solamente había realizado programas para ordenadores personales. Por otro lado, he empezado a estudiar el tratamiento digital de imágenes. En esto último, cabe destacar la importancia que OpenCV ha tenido para el desarrollo de la aplicación.

El desarrollo de este trabajo comienza en el mes de febrero, estudiando la aplicación AntibiogramJ. Hasta la finalización del mismo han ido surgiendo muchos problemas que me han hecho adquirir más experiencia. Uno de los errores que cometí, debido a mi inexperiencia en este campo, fue realizar una planificación demasiado optimista. Pensé que muchas de las tareas eran triviales y que no requerían para su realización mucho tiempo. A medida que pasaba el tiempo, he ido viendo como algunos de los objetivos propuestos no iban a poder ser alcanzados. Sin embargo, al mismo tiempo he podido comprobar como la aplicación que trataba de desarrollar iba cogiendo forma.

A finales de marzo el proyecto parecía no avanzar, ya que la detección del primer círculo resultó ser mucho más compleja de lo esperado. Sin embargo, una vez había detectado el primer círculo la motivación por seguir adelante creció exponencialmente. Estoy satisfecho con el trabajo realizado pese a no haber sido capaz de cumplir al 100% con la planificación inicial. Mi inexperiencia a la hora de planificar, me llevo a omitir algunas tareas importantes y a ser demasiado optimista con los plazos marcados.

Por otro lado, se me ha quedado la sensación de que para mejorar el algoritmo usado para detectar los discos de antibiótico a través de la búsqueda de zonas blancas, sería necesario un preprocesamiento de la imagen más intenso. Además, me he quedado con las ganas de terminar el nuevo algoritmo para la detección de halos.

Para poder desarrollar todo este trabajo ha sido indispensable hacer uso de gran parte de los conocimientos adquiridos a lo largo de toda la carrera. Si bien es cierto que he trabajado con tecnología desconocida para mí, la base de conocimiento que había adquirido en estos 4 años me ha permitido manejarme con esta tecnología de forma adecuada.

Finalmente, este trabajo me ha permitido darme cuenta de muchas de las cualidades que un ingeniero informático debe tener. Planificar eficientemente el trabajo a realizar, comunicarse adecuadamente para entender lo que el cliente quiere o ser capaz de aprender autónomamente son solo algunas de ellas.

Bibliografía

[1] Documentación oficial de OpenCV, http://opencv.org/

[2] Documentación oficial de Android: Android Developer Guide, https://developer.android.com

[3] Documentación oficial de AntibiogramJ, https://sourceforge.net/p/antibiogramj

[4] Documentación oficial de ImageJ, https://imagej.net/Development

[5] Documentación oficial de SQLite, https://www.sqlite.org/about.html

[6] Código Alonso. Tutoriales de ayuda de SQLite, https://www.youtube.com/channel/UCrjb6hMiNztfhH5mW-RBe4A

[7] Separación en capas Android: http://slashmobility.com/blog/2016/01/codigo-limpio-en-arquitecturas-android/

[8] Tutoriales de uso de OpenCV: http://acodigo.blogspot.com.es/p/tutorial-opencv.html

[9] JavaCV, https://github.com/bytedeco/javacv

[10] Ayuda para la detección de círculos utilizando JavaCV: http://opencvlover.blogspot.com.es/2012/07/hough-circle-in-javacv.html

[11] Página oficial de EUCAST: http://www.eucast.org/